



User Guide

AWS App Studio



AWS App Studio: User Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS App Studio?	1
Are you a first-time App Studio user?	1
Concepts	2
Administrator (Admin)	2
Application (App)	2
Automation	3
Automation actions	3
Builder	3
Application studio	3
Component	4
Entity	4
Connector	4
Page	4
Trigger	4
Setting up and signing in to App Studio	5
Creating and setting up an App Studio instance for the first time	5
Sign up for an AWS account	5
Create an administrative user for managing AWS resources	6
Enable App Studio from the AWS Management Console	6
Accepting an invitation to join App Studio	9
Getting started	10
Tutorial: Generate an app using AI	10
Prerequisites	11
Step 1: Create an application	11
Step 2: Explore your new application	12
Step 3: Preview your application	14
Next steps	14
Tutorial: Start building from an empty app	15
Prerequisites	17
Step 1: Create an application	18
Step 2: Create an entity to define your app's data	18
Step 3: Design the user interface (UI) and logic	21
Step 4: Preview the application	24
Step 5: Publish the application to the Testing environment	24

Next steps	25
Administrator documentation	26
Managing user access with groups and roles	26
Roles and permissions	26
Viewing groups	27
Adding users or groups	27
Changing a group's role	28
Removing users or groups	29
Connect to other services with connectors	30
Connect to AWS services	30
Connect to third-party services	52
Viewing, editing, and deleting connectors	59
Builder documentation	61
Creating, editing, and deleting applications	61
Viewing applications	61
Creating an application	62
Editing an application	63
Deleting an application	64
Previewing, publishing, and sharing applications	64
Previewing applications	65
Publishing applications	65
Sharing published applications	70
Building your app's user interface	71
Creating, editing, or deleting pages	71
Adding, editing, and deleting components	73
Configuring role-based visibility of pages	74
Components reference	76
Defining your app's business logic with automations	122
Automations concepts	122
Tutorial: Interacting with Amazon S3 using automations	124
Creating, editing, and deleting automations	133
Adding, editing, and deleting automation actions	135
Automation actions reference	136
Configure your app's data model with entities	153
Creating an entity	153
Configuring an entity	157

Deleting an entity	162
Managed data entities	162
Page and automation parameters	164
Page parameters	164
Automation parameters	165
Generative AI in App Studio	171
Generating your app	171
Generating your data models	171
Generating sample data	172
Configuring actions for AWS services	172
Mocking responses	172
Asking AI for help	172
Using JavaScript to write expressions	172
Basic syntax	173
Interpolation	173
Concatenation	173
Date and time	173
Code blocks	174
Global variables and functions	174
Accessing UI component values	174
Working with table data	176
Accessing automations	177
Troubleshooting and debugging apps	180
Troubleshooting in the application studio	180
Troubleshooting while previewing an app	180
Troubleshooting in the Testing environment	180
Debugging with logs from published apps in Amazon CloudWatch Logs	181
Troubleshooting connector issues	182
Building an app with multiple users	183
Invite builders to edit an app	183
Attempting to edit an app that is being edited by another user	184
Security	185
Security considerations and mitigations	185
Security considerations	185
Security risk mitigation recommendations	186
Service-linked roles	186

Service-linked role permissions for App Studio	187
Creating a service-linked role for App Studio	187
Editing a service-linked role for App Studio	188
Deleting a service-linked role for App Studio	188
AWS managed policies	188
AppStudioServiceRolePolicy	189
Policy updates	190
Document history	191

What is AWS App Studio?

AWS App Studio is a generative AI-powered service that uses natural language to create enterprise-grade applications. App Studio opens up application development to technical professionals without software development skills, such as IT project managers, data engineers, and enterprise architects, empowering them to quickly build applications that are secure and fully managed by AWS, eliminating the need for operational expertise. Builders can create and deploy apps to modernize internal business processes such as inventory management and tracking, claims processing, and complex approvals to improve employee productivity and customer outcomes.

Topics

- [Are you a first-time App Studio user?](#)

Are you a first-time App Studio user?

If you are a first-time user of App Studio, we recommend that you begin by reading the following sections:

- For users with the administrator role who will be setting up App Studio, managing users and access, and configuring connectors with other AWS or third-party services, see [AWS App Studio concepts](#) and [Setting up and signing in to AWS App Studio](#).
- For builders who will be creating and developing applications, see [AWS App Studio concepts](#) and [Getting started with AWS App Studio](#).

AWS App Studio concepts

Get familiar with the key concepts to help speed up creating applications and automating processes for your team. These concepts include terms used throughout App Studio for both administrators and builders.

Topics

- [Administrator \(Admin\)](#)
- [Application \(App\)](#)
- [Automation](#)
- [Automation actions](#)
- [Builder](#)
- [Application studio](#)
- [Component](#)
- [Entity](#)
- [Connector](#)
- [Page](#)
- [Trigger](#)

Administrator (Admin)

Admin is a role that can be assigned to a group in App Studio. Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.

Only admins have access to the **Admin Hub**, which contains tools to manage roles, data sources, and applications.

Application (App)

An **application** is a single configuration of pages that include assets such as interface components, automations, and data sources with which users can interact.

Automation

Built in the application studio, **automations** are how you define the business logic of your application. The main components of an automation are: triggers that start the automation, a sequence of one or more actions, input parameters used to pass data to the automation, and an output.

Automation actions

An automation action, commonly referred to as an **action**, is an individual step of logic that make up an automation. Each action performs a specific task, whether it's sending an email, creating a data record, invoking a Lambda function, or calling APIs. Actions are added to automations from the action library, and can be grouped into conditional statements or loops.

Builder

Builder is a role that can be assigned to a group in App Studio. Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.

Users with the Builder role have access to the **Builder Hub**, which contains details about resources such as the applications that the builder has access to along with helpful information such as learning resources.

Application studio

The **application studio** is a visual tool to build applications. This application studio includes the following tabs for building apps:

- Pages: Where builders design their applications with [pages](#) and [components](#).
- Automations: Where builders design their application's business logic with [automations](#).
- Data: Where builders design their application's data model with [entities](#).

The application studio also contains a debug console, and an AI chat window to get contextual help while building.

Component

Components are individual functional items within the UI of your application. Components are contained in pages, and some components can serve as a container for other components. Components combine UI elements with the business logic you want that UI element to perform. For example, one type of component is a form, where users can enter information in fields and, once submitted, that information is added as a database record.

Entity

Entities are data tables in App Studio. Entities interact directly with tables in data sources. Entities include fields to describe the data in them, queries to locate and return data, and mapping to connect the entity's fields to a data source's columns.

Connector

A **connector** is a connection between App Studio and other AWS services, such as AWS Lambda and Amazon Redshift, or third-party services. Once a connector is created and configured, builders can use it and the resources it connects to App Studio in their applications.

Only users with the Admin role can create, manage, or delete connectors.

Page

Pages are containers for [components](#), which make up the UI of an application in App Studio. Each page represents a screen of your application's user interface (UI) that your users will interact with. Pages are created and edited in the **Pages** tab of the application studio.

Trigger

Trigger determine when, and on what conditions, an automation will run. Some examples of triggers are On click for buttons and On select for text inputs. The type of component determines the list of available triggers for that component. Triggers are added to [components](#) and configured in the application studio.

Setting up and signing in to AWS App Studio

Setting up AWS App Studio is different depending on your role:

- **First-time setup as an AWS or organization administrator:** Setting up App Studio for the first time as an administrator includes creating an AWS account if you don't have one, creating the App Studio instance, and configuring user access using IAM Identity Center groups. After the instance is created, anyone with the Administrator role in App Studio can do further setting up tasks, such as configuring connectors to connect other services, such as data sources, to your App Studio instance. For information about first-time setup, see [Creating and setting up an App Studio instance for the first time](#).
- **Getting started as a builder:** When you receive an invitation to join App Studio as a builder, you must accept the invitation and activate your IAM Identity Center user credentials by providing a password. Afterwards, you can sign into App Studio and start building applications. For information about accepting an invitation and joining an App Studio instance, see [Accepting an invitation to join App Studio](#).

Creating and setting up an App Studio instance for the first time

Sign up for an AWS account

An AWS account is required to set up App Studio. Only one AWS account is required to use App Studio—builders and administrators do not need an AWS account to use App Studio, as access is managed with AWS IAM Identity Center.

To create an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign

administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

Create an administrative user for managing AWS resources

When you first create an AWS account, you begin with a default set of credentials with complete access to all AWS resources in your account. This identity is called the [AWS account root user](#). For creating AWS roles and resources to be used with App Studio, we strongly recommend you do not use the AWS account root user, and instead create and use an administrative user.

Use the following topics to create an administrative user for managing AWS roles and resources for use with App Studio.

- For a single, standalone AWS account, see [Create your first IAM user](#) in the *IAM User Guide*. You can provide any user name, but it must have AdministratorAccess permissions policy.
- For multiple AWS accounts managed through AWS Organizations, see [Set up AWS account access for an IAM Identity Center administrative user](#) in the *AWS IAM Identity Center User Guide*.

Enable App Studio from the AWS Management Console

To use App Studio, you must enable it from the App Studio landing page in the AWS Management Console.

To enable App Studio from the AWS Management Console


1. Open the App Studio console at <https://console.aws.amazon.com/appstudio/>.
2. Choose **Get started**.
3. The steps to set up App Studio are determined by whether or not you have an IAM Identity Center instance, and the type of instance.

Tip

To find more information about IAM Identity Center instances, including the different types and how to find which type you have, see [Manage organization and account instances of IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

- If you have an organization instance of IAM Identity Center:
 - a. In **Configure access to App Studio with Single Sign-On**, select existing IAM Identity Center groups to provide them with access to App Studio. App Studio groups will be created based on the specified configuration. Members of groups added to **Admin groups** will have the **Admin** role, and members of groups added to **Builder groups** will have the **Builder** role in App Studio. The roles are defined as follows:
 - Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.
 - Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.
 - b. In **Create Amazon CodeCatalyst space**, provide a name for the CodeCatalyst space that will be used to store App Studio source code and other information.
- If you have an account instance of IAM Identity Center instance:
 - a. In **Account permissions**, review the required permissions for enabling App Studio. If your account does not have the required permissions, you will not be able to enable App Studio. You must either get the required permissions added to your account, or switch to an account that has them.
 - b. In **Configure access to App Studio with Single Sign-On**, in **IAM Identity Center account**, choose **Use an existing account instance**
 - c. In **AWS Region**, choose the region in which your AWS Identity Center account instance is located.
 - d. Select existing IAM Identity Center groups to provide them with access to App Studio. App Studio groups will be created based on the specified configuration. Members of groups added to **Admin groups** will have the **Admin** role, and members of groups added to **Builder groups** will have the **Builder** role in App Studio. The roles are defined as follows:
 - Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.

- Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.
- If you do not have an IAM Identity Center instance:

 **Note**

Setting up App Studio automatically creates an IAM Identity Center account instance with the groups you configure during the set up process. After the setup is complete, you can add or manage users and groups in the IAM Identity Center console at <https://console.aws.amazon.com/singlesignon/>.

- a. In **Account permissions**, review the required permissions for enabling App Studio. If your account does not have the required permissions, you will not be able to enable App Studio. You must either get the required permissions added to your account, or switch to an account that has them.
- b. In **Configure access to App Studio with Single Sign-On**, in **IAM Identity Center account**, choose **Create an account instance for me**.
- c. In **Create users and groups and add them to App Studio**, provide a name for and add users to an admin group and builder group. Users added to the admin group will have the **Admin** role in App Studio, and users added to the builder group will have the **Builder** role. The roles are defined as follows:
 - Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.
 - Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.

 **Important**

You must add yourself as a user of the admin group to set up App Studio and have admin access after setting up.

4. In **Acknowledgement**, acknowledge the statements by choosing their checkboxes.
5. Choose **Set up**.

Accepting an invitation to join App Studio

Access to App Studio is managed by IAM Identity Center, so each user that wants to use App Studio must configure a user in IAM Identity Center and belong to a group that has been added to App Studio by an admin. When an administrator invites you to join IAM Identity Center, you'll receive an email asking you to accept the invitation and activate your user credentials. Once activated, you can use those credentials to sign in to App Studio.

To accept an invitation to IAM Identity Center to access App Studio

1. When you receive an invitation email, follow the steps to provide a password and activate your user credentials in IAM Identity Center. For more information, see [Accepting the invitation to join IAM Identity Center](#).
2. After you activate your user credentials, use them to sign into your App Studio instance.

Getting started with AWS App Studio

The following getting started tutorials walk you through building your first application in App Studio.

- (Recommended): To use generative AI to describe the app you want to create, and automatically create it and its resources, see [Tutorial: Generate an app using AI](#).
- To start building from an empty app, see [Tutorial: Start building from an empty app](#).

Tutorial: Generate an app using AI

App Studio contains generative AI features throughout the service to help speed up application building. In this tutorial, you'll learn how to generate an app using AI by describing your app using natural language. Using AI to generate an app is a great way to start building, as many of the app's resources are created for you and it's typically much easier to start building from a generated app with existing resources than starting from an empty app.

Note

You can view the [Build enterprise-grade applications with natural language using AWS App Studio \(preview\)](#) blog post to view a similar walkthrough that includes images. The blog post also contains information about setting up and configuring administrator-related resources, but you can skip to the portion about building applications if desired.

When an app is generated with AI, the app is created with the following resources that are tailored to the app you described:

1. **Pages and components:** *Components* are the building blocks of your application's user interface. They represent visual elements like tables, forms, buttons, and more. Each component has its own set of properties and can be customized to fit your specific requirements. Components are contained in *pages*.
2. **Automations:** *Automations* are used to define the logic and workflows that govern how your application behaves. Automations can create, update, read, or delete rows in a data table or objects in an Amazon S3 bucket, or handle tasks like data validation, notifications, or integrations with other systems.

3. **Entities:** Data is the information that powers your application. The generated app will create *entities*, similar to tables, that represent the different types of data you need to store and work with, such as customers, products, or orders. You can connect these data models to a variety of data sources, including AWS services and external APIs, using App Studio's connectors.

Contents

- [Prerequisites](#)
- [Step 1: Create an application](#)
- [Step 2: Explore your new application](#)
 - [Explore pages and components](#)
 - [Explore automations and actions](#)
 - [Explore data with entities](#)
- [Step 3: Preview your application](#)
- [Next steps](#)

Prerequisites

Before you get started, review and complete the following prerequisites:

- Access to AWS App Studio.
- Optional: Review [AWS App Studio concepts](#) to familiarize yourself with important App Studio concepts.

Step 1: Create an application

The first step in generating an app is to describe the app you want to create to the AI assistant in App Studio. You can review the application that will be generated, and iterate as desired before generating.

To generate your app using AI

1. Log in to App Studio.
2. Navigate to the builder hub and choose **+ Create app**.
3. Choose **Generate an app with AI**.

4. In the **App name** field, provide a name for your app.
5. In the **Select data sources** dialog box, choose **Skip**.
6. You can start defining the app you want to generate by describing it in the text box, or by choosing **Customize** on a sample prompt. Once you describe your app, App Studio will generate the app requirements and details for you to review. This includes use cases, user flows, and data models.
7. Use the text box to iterate your app as needed, until you are satisfied with the requirements and details.
8. Once you're ready to generate your app and start building, choose **Generate app**.
9. Optionally, you can view a short video that details how to navigate around your new app.
10. Choose **Edit app** to enter the application studio for your app.

Step 2: Explore your new application

In the application studio, you'll find the following resources:

1. A canvas that you use to view or edit your application. The canvas changes depending on the resource that is selected.
2. Navigation tabs at the top of the canvas. The tabs are described in the following list:
 - a. **Pages**: Where you use pages and components to design the UI of your app.
 - b. **Automations**: Where you use actions in automations to define the business logic of your app.
 - c. **Data**: Where you define entities, their fields, sample data, and data actions to define the data models of your app.
 - d. **App settings**: Where you define settings for your app, including app roles, which are used to define role-based visibility of pages for your end users.
3. A left-side navigation menu, which contains resources based on which tab you're viewing.
4. A right-side menu that lists resources and properties for selected resources in the **Pages** and **Automations** tabs.
5. A debug console that displays warnings and errors at the bottom of the builder. There may be errors present in your generated app. This is likely due to automations that require a configured connector to perform actions, such as sending an email with Amazon Simple Email Service.
6. An **Ask AI** chat window to get contextual help from the AI builder assistant.

Let's take a closer look at the **Pages**, **Automations**, and **Data** tabs.

Explore pages and components

In the **Pages** tab, you'll find pages and their components that were generated for you.

Each page represents a screen of your application's user interface (UI) that your users will interact with. Within these pages, you can find various components such as tables, forms, buttons, and more to create the desired layout and functionality.

Take some time to view the pages and their components by using the left-side navigation menu. When you select a page or component, you can choose **Properties** in the right-

Explore automations and actions

In the **Automations** tab, you'll find automations and their actions that were generated for you.

Automations define the business logic of your app, such as creating, viewing, updating, or deleting data entries, sending emails, or even invoking APIs or Lambda functions.

Take some time to view the automations by using the left-side navigation menu. When you choose an automation, you can view its properties in the right-side **Properties** menu. An automation contains the following resources:

1. Automations are made up of individual actions, which are the building blocks of your app's business logic. You can view the actions of an automation in the left-side navigation menu, or in the canvas of a selected automation. When you select an action, you can view its properties in the right-side **Properties** menu.
2. Automation parameters are how data is passed into an automation. Parameters act as placeholders that are replaced with actual values when the automation is run, allowing you to use the same automation with different inputs each time.
3. Automation output is where you configure the result of an automation. By default, an automation has no output, so to use an automation's result in components or other automations, you must define them here.

For more information, see [Automations concepts](#).

Explore data with entities

In the **Data** tab, you'll find entities that were generated for you.

Entities represent tables that hold your application's data, similar to tables in a database. Instead of connecting your application's user interface (UI) and automations directly to data sources, they connect to entities first. Entities act as a middle-person between your actual data source and your App Studio app, providing a single place to manage and access your data.

Take some time to view the entities that were generated by choosing them from the left-side navigation menu to see the following details:

1. In the **Configuration** tab, you'll find the entity name, and its fields, which represent the columns of your entity.
2. In the **Data actions** tab, you'll find the data actions generated with your entity. Components and automations can use data actions to fetch data from your entity.
3. In the **Sample data** tab, you'll find sample data which can be used to test your app in the Development environment, which does not communicate with external services. For more information about environments, see [Application environments](#).
4. In the **Connection** tab, you'll see information about the external data sources that the entity is connected to. App Studio provides a managed data storage solution that uses a DynamoDB table. For more information, see [Managed data entities in AWS App Studio](#).

Step 3: Preview your application

You can preview applications in App Studio to see how they will appear to users and also test its functionality by using it and checking logs in a debug panel.

The application preview environment does not support displaying live data or the connection with external resources with connectors, such as data sources. Instead, you can use sample data and mocked output to test functionality.

To preview your app for testing

1. In the top-right corner of the app builder, choose **Preview**.
2. Interact with the pages of your app.

Next steps

Now that you've created your first app, here are some next steps:

1. If you haven't, read the [Build enterprise-grade applications with natural language using AWS App Studio \(preview\)](#) blog post for another getting started walkthrough that includes images.
2. Apps use connectors to send and receive data or communicate with external services, both AWS services and third-party services. It's necessary to learn more about connectors and how to configure them to build apps. Note that you must have the **Admin** role to manage connectors. Check out [Connect App Studio to other services with connectors](#) to learn more.
3. To learn more about previewing, publishing, and eventually sharing your app to end users, check out [Previewing, publishing, and sharing applications](#).
4. Keep exploring and updating the app you generated for some hands-on experience.
5. Check out the [Builder documentation](#) to learn more about building apps. Specifically, the following topics might be useful to explore:
 - [Automation actions reference](#)
 - [Components reference](#)
 - [Tutorial: Interacting with Amazon Simple Storage Service using automations](#)
 - [Security considerations and mitigations](#)

Tutorial: Start building from an empty app

In this tutorial, you'll build an internal Customer Meeting Request application using AWS App Studio. You'll learn about how to build apps in App Studio, including defining data structures, UI design, and app deployment, while focusing on real-world use cases and hands-on examples.

Note

This tutorial details how to build an app from scratch, starting with an empty app. Typically, it's much quicker and easier to use AI to help generate an app and its resources for you by providing a description of the app you want to create. For more information, see [Tutorial: Generate an app using AI](#).

The key to understanding how to build applications with App Studio is to understand the following four core concepts and how they work together: **components**, **automations**, **data**, and **connectors**.

1. **Components:** Components are the building blocks of your application's user interface. They represent visual elements like tables, forms, buttons, and more. Each component has its own set of properties and can be customized to fit your specific requirements.
2. **Automations:** Automations allows you to define the logic and workflows that govern how your application behaves. You can create automations to create, update, read, or delete rows in a data table or objects in an Amazon S3 bucket, or handle tasks like data validation, notifications, or integrations with other systems.
3. **Data:** Data is the information that powers your application. In App Studio, you can define data models, called *entities*, that represent the different types of data you need to store and work with, such as customer meeting requests, agenda, or attendees. You can connect these data models to a variety of data sources, including AWS services and external APIs, using App Studio's connectors.
4. **Connectors:** App Studio provides seamless connections with a wide range of data sources, including AWS services such as Aurora, DynamoDB, and Amazon Redshift, and third-party services such as Salesforce or many others using OpenAPI or generic API connectors. You can use App Studio's connectors to easily incorporate data and functionality from these enterprise-grade services and external applications into your applications.

As you progress through the tutorial, you'll explore how the key concepts of components, data, and automation come together to build your internal Customer Meeting Request application.

1. **Start with data:** Many applications begin with a data model, so this tutorial begins with data as well. To build the Customer Meeting Request app, you'll start by creating a `MeetingRequests` entity. This entity represents the data structure for storing all the relevant meeting request information, such as customer name, meeting date, agenda, and attendees. This data model serves as the foundation for your application, powering the various components and automations you'll build.
2. **Create your user interface (UI):** With the data model in place, the tutorial will then guide you through building the user interface (UI). In App Studio, the UI is built by adding *pages*, and adding *components* to them. You'll add components like *Tables*, *Detail views*, and *Calendars* to a meeting request dashboard page. These components will be designed to display and interact with the data stored in the `MeetingRequests` entity, allowing your users to view, manage, and schedule customer meetings. You will also create a meeting request creation page, which includes a *Form* component to collect data and a *Button* component to submit it.

- 3. Add business logic with automations:** To enhance the functionality of your application, you'll configure some of the components to enable user interactions, such as navigating to a page or creating a new meeting request record in the `MeetingRequests` entity.
- 4. Enhance with validation and expressions:** To ensure the integrity and accuracy of your data, you'll add validation rules to the *Form* component. This will help guarantee that users provide complete and valid information when creating new meeting request records. Additionally, you'll use expressions to reference and manipulate data within your application to display dynamic and contextual information throughout your user interface.
- 5. Preview and test:** Before deploying your application, you'll have the opportunity to preview and test it thoroughly. This will allow you to verify that the components, data, and automations are all working together seamlessly, providing your users with a smooth and intuitive experience.
- 6. Publish the application:** Finally, you'll deploy your completed internal Customer Meeting Request application, making it accessible to your users. With the power of App Studio's low-code approach, you'll have built a custom application that meets the specific needs of your organization, without the need for extensive programming expertise.

Contents

- [Prerequisites](#)
- [Step 1: Create an application](#)
- [Step 2: Create an entity to define your app's data](#)
 - [Add fields to your entity](#)
- [Step 3: Design the user interface \(UI\) and logic](#)
 - [Add a meeting request dashboard page](#)
 - [Add a meeting request creation page](#)
- [Step 4: Preview the application](#)
- [Step 5: Publish the application to the Testing environment](#)
- [Next steps](#)

Prerequisites

Before you get started, review and complete the following prerequisites:

- Access to AWS App Studio.

- Optional: Review [AWS App Studio concepts](#) to familiarize yourself with important App Studio concepts.
- Optional: An understanding of basic web development concepts, such as JavaScript syntax.
- Optional: Familiarity with AWS services.

Step 1: Create an application

1. Log in to App Studio.
2. Navigate to the builder hub and choose **+ Create app**.
3. Choose **Start from scratch**.
4. In the **App name** field, provide a name for your app, such as **Customer Meeting Requests**.
5. If asked to select data sources or a connector, choose **Skip** for the purposes of this tutorial.
6. Choose **Next** to proceed.
7. (Optional): Watch the video tutorial for a quick overview of building apps in App Studio.
8. Choose **Edit app** which will bring you into the App Studio app builder.

Step 2: Create an entity to define your app's data

Entities represent tables that hold your application's data, similar to tables in a database. Instead of connecting your application's user interface (UI) and automations directly to data sources, they connect to entities first. Entities act as a middle-person between your actual data source and your App Studio app, providing a single place to manage and access your data.

There are four ways to create an entity– for this tutorial, you will use the App Studio managed entity. Creating a managed entity also creates a corresponding DynamoDB table that is managed by App Studio. When changes are made to the entity in the App Studio app, the DynamoDB table is updated automatically. With this option, you don't have to manually create, manage, or connect a third-party data source, or designate mapping from entity fields to table columns.

When creating an entity, you must define a **primary key** field. A primary key serves as a unique identifier for each record or row in the entity, ensuring that each record can be easily identified and retrieved without ambiguity. The primary key consists of the following properties:

- Primary key name: A name for the primary key field of the entity.

- **Primary key data type:** The type of the primary key field. In App Studio, supported primary key types are **String** for text, and **Float** for a number. A text primary key, like *meetingName*, would have a type of **String** and a numerical primary key, such as *meetingId* would have a type of **Float**.

The primary key is a crucial component of an entity because it enforces data integrity, prevents data duplication, and enables efficient data retrieval and querying.

To create a managed entity

1. Choose **Data** from the top bar menu.
2. Choose **+ Create entity**.
3. Choose **Create App Studio managed entity**.
4. In the **Entity name** field, provide a name for your entity. For this tutorial, enter **MeetingRequests**.
5. In the **Primary key** field enter the primary key name label to give to the primary key column in your entity. For this tutorial, enter **requestID**.
6. In the **Primary key data type** dropdown, select **Float**.
7. Choose **Create entity**.

Add fields to your entity

For each field, you will specify the **display name**, which is the label visible to app users. The display name can contain spaces and special characters but must be unique within the entity. The display name serves as a user-friendly label for the field, and helps users easily identify and understand its purpose.

Next, you'll provide the **system name**, a unique identifier used internally by the application to reference the field. The system name should be concise and without spaces or special characters. The system name allows the application to make changes to the field's data. It acts as a unique reference point for the field within the application.

Finally, you'll select the **data type** that best represents the kind of data you want to store in the field, such as String (text), Boolean (true/false), Date, Decimal, Float, Integer, or DateTime. Defining the appropriate data type ensures data integrity and enables proper handling and processing

of the field's values. For instance, if you're storing customer names in your meeting request, you would select the `String` data type to accommodate text values.

To add fields to your `MeetingRequests` entity

- Choose **+ Add field** to add the five following fields:
 - a. Add a field that represents a customer's name with the following information:
 - **Display name:** `Customer name`
 - **System name:** `customerName`
 - **Data type:** `String`
 - b. Add a field that represents the meeting date with the following information:
 - **Display name:** `Meeting date`
 - **System name:** `meetingDate`
 - **Data type:** `DateTime`
 - c. Add a field that represents the meeting agenda with the following information:
 - **Display name:** `Agenda`
 - **System name:** `agenda`
 - **Data type:** `String`
 - d. Add a field to represent the meeting attendees with the following information:
 - **Display name:** `Attendees`
 - **System name:** `attendees`
 - **Data type:** `String`

You can add sample data to your entity, which can be used to test and preview your application before publishing it. By adding up to 500 rows of mock data, you can simulate real-world scenarios and examine how your application handles and displays various types of data, without relying on actual data or connecting to external services. This allows you to identify and resolve any issues or inconsistencies early in the development process, ensuring that your application functions as intended when dealing with actual data.

To add sample data to your entity.

1. Choose the **Sample data** tab in the banner.
2. Choose **Generate more sample data**.
3. Choose **Save**.

Optionally, choose **Connection** in the banner to review the details about the connector and DynamoDB table created for you.

Step 3: Design the user interface (UI) and logic

Add a meeting request dashboard page

In App Studio, each page represents a screen of your application's user interface (UI) that your users will interact with. Within these pages, you can add various components such as tables, forms, buttons, and more to create the desired layout and functionality.

Newly created applications come with a default page, so you will rename that one instead of adding a new one to use as a simple meeting request dashboard page.

To rename the default page


1. In the top bar navigation menu, choose **Pages**.
2. In the left-side panel, double click **Page1** and rename it to **MeetingRequestsDashboard**. Press enter.

Now, add a table component to the page that will be used to display meeting requests.

To add a table component to the meeting requests dashboard page

1. In the right-hand **Components** panel, locate the **Table** component and drag it onto the canvas.
2. Choose the table in the canvas to select it.
3. In the right-side **Properties** panel, update the following settings:
 - a. Choose the pencil icon to rename the table to **meetingRequestsTable**.
 - b. In the **Source** dropdown, choose **Entity**.
 - c. In the **Data actions** dropdown, choose the entity you created (**MeetingRequests**) and choose **+ Add data actions**.

4. If prompted, choose **getAll**.

 **Note**

The **getAll** data action is a specific type of data action that retrieves all the records (rows) from a specified entity. When you associate the getAll data action with a table component, for example, the table will automatically populate with all the data from the connected entity, displaying each record as a row in the table.

Add a meeting request creation page

Next, create a page that contains a form that end users use to create meeting requests. Along with the form, you will also add a submit button that creates the record in the MeetingRequests entity, and then navigates the end user back to the MeetingRequestsDashboard page.

Add a meeting request creation page

1. In the top banner, choose **Pages**.
2. In the left-side panel, choose **+ Add**.
3. In the right-side properties panel, select the pencil icon and rename the page to **CreateMeetingRequest**.

Now that the page is added, you will add a form to the page that end users will use to input information to create a meeting request in the MeetingRequests entity. App Studio offers a method of generating a form from an existing entity, which autopopulates the form fields based on the entity's fields and also generates a submit button for creating a record in the entity with the form inputs.

To automatically generate a form from an entity on the meeting request creation page

1. In the right-side **Components** menu, find the **Form** component and drag it onto the canvas.
2. Select **Generate form**.
3. From the dropdown, select the MeetingRequests entity.
4. Choose **Generate**.
5. Choose the **Submit** button on the canvas to select it.
6. In the right-side properties panel, in the **Triggers** section, choose **+ Add**.

7. Choose **Navigate**.
8. In the right-side properties panel, change the **Action name** to something descriptive, such as **Navigate to MeetingRequestsDashboard**.
9. Change the **Navigation type** to page. In the **Navigate to** dropdown, choose **MeetingRequestsDashboard**.

Now we have a meeting request creation page and form, and we want to make it easy to navigate to this page from the MeetingRequestsDashboard page, so that end users reviewing the dashboard can easily create meeting requests. Use the following procedure to create a button on the MeetingRequestsDashboard page that navigates to the CreateMeetingRequest page.

To add a button to navigate from MeetingRequestsDashboard to CreateMeetingRequest

1. In the top banner, choose **Pages**.
2. Choose the MeetingRequestsDashboard page.
3. In the right-side **Components** panel, find the **Button** component and drag it onto the canvas, placing it above the table.
4. Choose the newly added button to select it.
5. In the right-side **Properties** panel, update the following settings:
 - a. Select the pencil icon to rename the button to **createMeetingRequestButton**.
 - b. **Button label: Create Meeting Request**. This is the name that end users will see.
 - c. In the **Icon** dropdown, select **+ Plus**.
 - d. Create a trigger that navigates the end user to the MeetingRequestsDashboard page:
 1. In the **Triggers** section, choose **+ Add**.
 2. In **Action Type**, select **Navigate**.
 3. Choose the trigger you just created to configure it.
 4. In **Action name**, provide a descriptive name such as **NavigateToCreateMeetingRequest**.
 5. In the **Navigate type** dropdown, select **Page**.
 6. In the **Navigate to** dropdown, select the CreateMeetingRequest page.

Step 4: Preview the application

You can preview applications in App Studio to see how they will appear to users and also test its functionality by using it and checking logs in a debug panel.

The application preview environment does not support displaying live data or the connection with external resources with connectors, such as data sources. Instead, you can use sample data and mocked output to test functionality.

To preview your app for testing

1. In the top-right corner of the app builder, choose **Preview**.
2. Interact with the `MeetingRequestsDashboard` page, testing the table, form, and buttons.

Step 5: Publish the application to the Testing environment

Now that you're done creating, configuring, and testing your application, it's time to publish it to the **Testing** environment to perform final testing and then share it with users.

To publish your app to the Testing environment

1. In the top-right corner of the app builder, choose **Publish**.
2. Add a version description for the Testing environment.
3. Review and select the checkbox regarding the SLA.
4. Choose **Start**. Publishing may take up to 15 minutes.
5. (Optional) Once ready, you can give others access by choosing **Share** modal and following the prompts.

Note

An admin must have created end-user groups to share apps.

After testing, choose **Publish** again to promote the application to the Production environment. For more information about the different application environments, see [Application environments](#).

Next steps

Now that you've created your first app, here are some next steps:

1. Keep building the tutorial app. Now that you have data, some pages, and an automation configured, you can add additional pages and add components to learn more about building apps.
2. Check out the [Builder documentation](#) to learn more about building apps. Specifically, the following topics might be useful to explore:
 - [Automation actions reference](#)
 - [Components reference](#)
 - [Tutorial: Interacting with Amazon Simple Storage Service using automations](#)
 - [Security considerations and mitigations](#)

In addition, the following topics contain more information about concepts discussed in the tutorial:

- [Previewing, publishing, and sharing applications](#)
- [Creating an entity in an App Studio app](#)

Administrator documentation

The following topics contain information to help users who are managing third-party service connections and access, users, and roles in App Studio.

Topics

- [Managing access and roles in App Studio](#)
- [Connect App Studio to other services with connectors](#)

Managing access and roles in App Studio

One of the responsibilities of administrators in App Studio is to manage access, roles, and permissions. The following topics contain information about the roles in App Studio, and how to add users, remove users, or change their role.

Access to AWS App Studio is managed using IAM Identity Center groups. To add users to your App Studio instance, you must either:

- Add them to an existing IAM Identity Center group that is added to App Studio.
- Add them to a new or existing IAM Identity Center group that is not added to App Studio, and then add it to App Studio.

Because roles are applied to groups, the IAM Identity Center groups should represent the access privileges (or roles) you want to assign to members of the group. For more information about IAM Identity Center, including information about managing users and groups, see the [IAM Identity Center User Guide](#).

Roles and permissions

There are three roles in App Studio. The following list contains each role and their description.

- **Admin:** Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.
- **Builder:** Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.

- **App User:** App Users can access and use published apps, but cannot access your App Studio instance to build apps or manage resources.

In App Studio, roles are assigned to groups, therefore each member of an added IAM Identity Center group will be assigned the role that is assigned to the group.

Viewing groups

Perform the following steps to view the groups added to your App Studio instance.

Note

You must be an Admin to view groups in your App Studio instance.

To view groups added to your App Studio instance

- In the navigation pane, choose **Roles** in the **Manage** section. You will be taken to a page displaying a list of existing groups as well as each group's assigned role.

For information about managing groups, see [Adding users or groups](#), [Changing a group's role](#), or [Removing users or groups from App Studio](#).

Adding users or groups

To add users to App Studio, you must add them to an IAM Identity Center group and add that group to App Studio. Perform the following steps to add users to App Studio by adding IAM Identity Center groups and assigning a role.

Note

You must be an Admin to add users to your App Studio instance.

To add users or groups to your App Studio instance

1. To add users to your App Studio instance, you must either add them to an existing IAM Identity Center group that has been added to App Studio, or create a new IAM Identity Center group, add the new user to it, and add the new group to App Studio.

For information about managing IAM Identity Center users and groups, see [Manage identities in IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. If you added users to an existing IAM Identity Center group that was already added to App Studio, the new user can access App Studio with the designated permissions after completing the setup of their IAM Identity Center permissions. If you created a new IAM Identity Center group, perform the following steps to add the group to App Studio and designate a role for the group's members.
3. In the navigation pane, choose **Roles** in the **Manage** section.
4. On the **Roles** page, choose **+ Add group**. This will open an **Add groups** dialog box for you to enter information about the group.
5. In the **Add groups** dialog box, enter the following information:
 - Choose the existing IAM Identity Center group in the dropdown.
 - Select a role for the group.
 - **Admin**: Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.
 - **Builder**: Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.
 - **App User**: App Users can access and use published apps, but cannot access your App Studio instance to build apps or manage resources.
6. Choose **Assign** to add the group to App Studio and provide its members with the configured role.

Changing a group's role

Follow these steps to change the role assigned to a group in App Studio. Changing a group's role will change the role of every member in that group.

Note

You must be an Admin to change the role of a group in App Studio.

To change the role of a group

1. In the navigation pane, choose **Roles** in the **Manage** section. You will be taken to a page displaying a list of existing groups as well as each group's assigned role.
2. Choose the ellipses icon (...) and choose **Change role**.
3. In the **Change role** dialog box, select a new role for the group:
 - **Administrator**: Admins can manage users and groups within App Studio, add and manage connectors, and manage applications created by builders. Additionally, users with the Admin role have all of the permissions included with the Builder role.
 - **Builder**: Builders can create and build applications. Builders cannot manage users or groups, add or edit connector instances, or manage other builders' applications.
 - **App User**: App Users can access and use published apps, but cannot access your App Studio instance to build apps or manage resources.
4. Choose **Change** change the group's role.

Removing users or groups from App Studio

You cannot remove an IAM Identity Center group from App Studio. Performing the following instructions will instead downgrade the group's role to **App User**. Members of the group will still be able to access published App Studio apps.

To remove all access to App Studio and its apps, you must either delete the IAM Identity Center group or users in the AWS IAM Identity Center console. For information about managing IAM Identity Center users and groups, see [Manage identities in IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

Note

You must be an Admin to downgrade a group's access in App Studio.

To remove a group

1. In the navigation pane, choose **Roles** in the **Manage** section. You will be taken to a page displaying a list of existing groups as well as each group's assigned role.
2. Choose the ellipses icon (...) and choose **Revoke role**.

3. In the **Revoke role** dialog box, choose **Revoke** to downgrade the group's role to **App User**.

Connect App Studio to other services with connectors

A **connector** is a connection between App Studio and other AWS services, such as AWS Lambda and Amazon Redshift, or third-party services. Once a connector is created and configured, builders can use it and the resources it connects to App Studio in their applications.

Only users with the Admin role can create, manage, or delete connectors.

Topics

- [Connect to AWS services](#)
- [Connect to third-party services](#)
- [Viewing, editing, and deleting connectors](#)

Connect to AWS services

Topics

- [Connect to Amazon Redshift](#)
- [Connect to Amazon DynamoDB](#)
- [Connect to AWS Lambda](#)
- [Connect to Amazon Simple Storage Service \(Amazon S3\)](#)
- [Connect to Amazon Aurora](#)
- [Connect to Amazon Bedrock](#)
- [Connect to AWS services using the Other AWS services connector](#)

Connect to Amazon Redshift

To connect App Studio with Amazon Redshift to enable builders to access and use Amazon Redshift resources in applications, you must perform the following steps:

1. [Create and configure Amazon Redshift resources](#)
2. [Create an IAM role to give App Studio access to Amazon Redshift resources](#)
3. [Create Amazon Redshift connector](#)

Create and configure Amazon Redshift resources

Use the following procedure to create and configure Amazon Redshift resources to be used with App Studio.

To set up Amazon Redshift for use with App Studio

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshiftv2/>.

We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).

2. Create a Redshift Serverless data warehouse or a provisioned cluster. For more information, see [Creating a data warehouse with Redshift Serverless](#) or [Creating a cluster](#) in the *Amazon Redshift User Guide*.
3. Once provisioning is complete, choose **Query Data** to open the query editor. Connect to your database.
4. Change the following settings:
 1. Set **Isolated session** toggle to OFF. This is needed so that you can see data changes made by other users, such as from a running App Studio application.
 2. Choose the "gear" icon. Choose **Account settings**. Increase **Maximum concurrent connections** to 10. This is the limit on the number of query editor sessions that can connect to a Amazon Redshift database. It does not apply to other clients such as App Studio applications.
5. Create your data tables under the `public` schema. INSERT any initial data into these tables.
6. Run the following commands in query editor:

```
CREATE USER "IAMR:AppBuilderDataAccessRole" WITH PASSWORD DISABLE;
```

```
GRANT ALL ON ALL TABLES IN SCHEMA public to "IAMR:AppBuilderDataAccessRole";
```

Create an IAM role to give App Studio access to Amazon Redshift resources

To use Amazon Redshift resources with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of data that

builders can use and what operations can be called against that data, such as Create, Read, Update, or Delete.

We recommend creating at least one IAM role per service and policy. For example, if builders are creating two applications backed by the same tables in Amazon Redshift, one that only requires read access, and one that requires read, create, update and delete; an administrator should create two IAM roles, one using read only permissions, and one with full CRUD permissions to the applicable tables in Amazon Redshift.

To create an IAM role to give App Studio access to Amazon Redshift resources

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/IsAppStudioAccessRole": "true"
        }
      }
    }
  ]
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the + next to a policy will expand the policy to show the permissions granted by it. For Amazon Redshift, you may consider adding the following policies:
 - **AmazonRedshiftFullAccess**: Grants full access to all Amazon Redshift resources. Additionally, this policy grants full access to all Redshift Serverless resources.
 - **AmazonRedshiftDataFullAccess**: Grants full access to the Redshift Data API operations and resources.

For more information about using IAM policies with Amazon Redshift, including a list of managed policies and their descriptions, see [Using identity-based policies \(IAM policies\) for Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key**: IsAppStudioDataAccessRole
 - **Value**: true
8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it when [creating the Amazon Redshift connector in App Studio](#).

Create Amazon Redshift connector

To create a connector for Amazon Redshift

1. In the navigation pane, choose **connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Add**.
3. Choose **AWS services** in the **AWS** section of the supported services list.
4. Choose **Next**.
5. Configure your connector by filling out the following fields:

- **Name:** Provide a name for your connector.
 - **Description:** Provide a description for your connector.
 - **IAM Role:** Enter the Amazon Resource Name (ARN) from the IAM role created in [Create an IAM role to give App Studio access to Amazon Redshift resources](#). For more information about IAM, see the [IAM User Guide](#).
 - **Region:** Choose the AWS Region where your Amazon Redshift resources are located.
 - **Compute type:** Choose if you are using Amazon Redshift Serverless or a provisioned cluster.
 - **Cluster or Workgroup selection:** If **Provisioned** is chosen, choose the cluster you want to connect to App Studio. If **Serverless** is chosen, choose the workgroup.
 - **Database selection:** Choose the database you want to connect to App Studio.
 - **Available tables:** Select the tables you want to connect to App Studio.
6. Choose **Next**. Review the connection information and choose **Create**.
 7. The newly created connector will appear in the **connectors** list.

Connect to Amazon DynamoDB

To connect App Studio with DynamoDB to enable builders to access and use DynamoDB resources in applications, you must perform the following steps:

1. [Create and configure DynamoDB resources](#)
2. [Create an IAM role to give App Studio access to DynamoDB resources](#)
3. [Create DynamoDB connector](#)

Create and configure DynamoDB resources

Use the following procedure to create and configure DynamoDB resources to be used with App Studio.

To set up DynamoDB for use with App Studio

1. Sign in to the AWS Management Console and open the DynamoDB console at <https://console.aws.amazon.com/dynamodb/>.

We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).

2. In the left navigation pane, choose **Tables**.
3. Choose **Create table**.
4. Enter a name and keys for your table.
5. Choose **Create table**.
6. After your table is created, add some items to it so they will appear once the table is connected to App Studio.
 - a. Choose your table, choose **Actions**, and choose **Explore items**.
 - b. In **Items returned**, choose **Create item**.
 - c. (Optional): Choose **Add new attribute** to add more attributes to your table.
 - d. Enter values for each attribute and choose **Create item**.

Create an IAM role to give App Studio access to DynamoDB resources

To use DynamoDB resources with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of data that builders can use and what operations can be called against that data, such as Create, Read, Update, or Delete.

We recommend creating at least one IAM role per service and policy. For example, if builders are creating two applications backed by the same tables in DynamoDB, one that only requires read access, and one that requires read, create, update and delete; an administrator should create two IAM roles, one using read only permissions, and one with full CRUD permissions to the applicable tables in DynamoDB.

To create an IAM role to give App Studio access to DynamoDB resources

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/IsAppStudioAccessRole": "true"
        }
      }
    }
  ]
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the + next to a policy will expand the policy to show the permissions granted by it. For DynamoDB, you may consider adding the `AmazonDynamoDBFullAccess` policy, which grants full access to the DynamoDB workgroups.

For more information about using IAM policies with DynamoDB, including a list of managed policies and their descriptions, see [Identity and Access Management for Amazon DynamoDB](#) in the *Amazon DynamoDB Developer Guide*.

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key:** `IsAppStudioDataAccessRole`
 - **Value:** `true`

8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it when [creating the DynamoDB connector in App Studio](#).

Create DynamoDB connector

To create a connector for DynamoDB

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **Amazon DynamoDB** from the list of connector types.
4. Configure your connector by filling out the following fields:
 - **Name:** Enter a name for your DynamoDB connector.
 - **Description:** Enter a description for your DynamoDB connector.
 - **IAM role:** Enter the Amazon Resource Name (ARN) from the IAM role created in [Create an IAM role to give App Studio access to DynamoDB resources](#). For more information about IAM, see the [IAM User Guide](#).
 - **Region:** Choose the AWS Region where your DynamoDB resources are located.
 - **Available tables:** Select the tables you want to connect to App Studio.
5. Choose **Next**. Review the connection information and choose **Create**.
6. The newly created connector will appear in the **Connectors** list.

Connect to AWS Lambda

To connect App Studio with Lambda to enable builders to access and use Lambda resources in applications, you must perform the following steps:

1. [Create and configure Lambda functions](#)
2. [Create an IAM role to give App Studio access to Lambda resources](#)
3. [Create Lambda connector](#)

Create and configure Lambda functions

If you don't have existing Lambda functions, you must first create them. To learn more about creating Lambda functions, see the [AWS Lambda Developer Guide](#).

Create an IAM role to give App Studio access to Lambda resources

To use Lambda resources with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of data that builders can use and what operations can be called against that data, such as Create, Read, Update, or Delete.

We recommend creating at least one IAM role per service and policy. For example, if builders are creating two applications backed by the same tables in Lambda, one that only requires read access, and one that requires read, create, update and delete; an administrator should create two IAM roles, one using read only permissions, and one with full CRUD permissions to the applicable tables in Lambda.

To create an IAM role to give App Studio access to Lambda resources

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
```

```
        "Condition": {
            "StringEquals": {
                "aws:PrincipalTag/IsAppStudioAccessRole": "true"
            }
        }
    ]
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the + next to a policy will expand the policy to show the permissions granted by it. For Lambda, you may consider adding the `AWSLambdaRole` policy, which grants permissions to invoke Lambda functions.

For more information about using IAM policies with Lambda, including a list of managed policies and their descriptions, see [Identity and Access Management for AWS Lambda](#) in the *AWS Lambda Developer Guide*.

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key:** `IsAppStudioDataAccessRole`
 - **Value:** `true`
8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it when [creating the Lambda connector in App Studio](#).

Create Lambda connector

To create a connector for Lambda

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **AWS Services** from the list of connector types.

4. Configure your connector by filling out the following fields:
 - **Name:** Enter a name for your Lambda connector.
 - **Description:** Enter a description for your Lambda connector.
 - **IAM role:** Enter the Amazon Resource Name (ARN) from the IAM role created in [Create an IAM role to give App Studio access to Lambda resources](#). For more information about IAM, see the [IAM User Guide](#).
 - **Service:** Choose **Lambda**.
 - **Region:** Choose the AWS Region where your Lambda resources are located.
5. Choose **Create**.
6. The newly created connector will appear in the **Connectors** list.

Connect to Amazon Simple Storage Service (Amazon S3)

To connect App Studio with Amazon S3 to enable builders to access and use Amazon S3 resources in applications, you must perform the following steps:

1. [Create and configure Amazon S3 resources](#)
2. [Create an IAM role to give App Studio access to Amazon S3 resources](#)
3. [Create Amazon S3 connector](#)

After you have completed the steps and created the connector with proper permissions, builders can use the connector to create apps that interact with Amazon S3 resources. For more information about interacting with Amazon S3 in App Studio apps, see [Tutorial: Interacting with Amazon Simple Storage Service using automations](#).

Create and configure Amazon S3 resources

Depending on your app's needs and your existing resources, you may need to create an Amazon S3 bucket for apps to write to and read from. For information about creating Amazon S3 resources, including buckets, see [Getting started with Amazon S3](#) in the *Amazon Simple Storage Service User Guide*.

To use the [S3 upload](#) component in your apps, you must add a cross-origin resource sharing (CORS) configuration to any Amazon S3 buckets you want to upload to. The CORS configuration gives App Studio permission to push objects to the bucket. The following procedure

details how to add a CORS configuration to an Amazon S3 bucket using the console. For more information about CORS and configuring it, see [Using cross-origin resource sharing \(CORS\)](#) in the *Amazon Simple Storage Service User Guide*.

To add a CORS configuration to an Amazon S3 bucket in the console

1. Navigate to your bucket in the <https://console.aws.amazon.com/s3/>.
2. Choose the **Permissions** tab.
3. In **Cross-origin resource sharing (CORS)**, choose **Edit**.
4. Add the following snippet:

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "PUT",
      "POST"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

5. Choose **Save changes**.

Create an IAM role to give App Studio access to Amazon S3 resources

To use Amazon S3 resources with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of data that builders can use and what operations can be called against that data, such as Create, Read, Update, or Delete.

We recommend creating at least one IAM role per service and policy.

To create an IAM role to give App Studio access to Amazon S3 resources

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/IsAppStudioAccessRole": "true"
        }
      }
    }
  ]
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the + next to a policy will expand the policy to show the permissions granted by it. Consider adding one of the following policies, based on your app's needs:
 - **AmazonS3FullAccess**: Grants permissions that allow full access to Amazon S3.
 - **AmazonS3ReadOnlyAccess**: Grants permissions that allow read-only access to Amazon S3.

For more information about using IAM policies with Amazon S3, including a list of managed policies and their descriptions, see [Identity and Access Management for Amazon Simple Storage Service](#) in the *Amazon Simple Storage Service User Guide*.

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key:** `IsAppStudioDataAccessRole`
 - **Value:** `true`
8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it to create the Amazon S3 connector in App Studio in the next step.

Create Amazon S3 connector

To create a connector for Amazon S3

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **AWS Services** from the list of connector types.
4. Configure your connector by filling out the following fields:
 - **Name:** Enter a name for your Amazon S3 connector.
 - **Description:** Enter a description for your Amazon S3 connector.
 - **IAM role:** Enter the Amazon Resource Name (ARN) from the IAM role created in [Create an IAM role to give App Studio access to Amazon S3 resources](#). For more information about IAM, see the [IAM User Guide](#).
 - **Region:** Choose the AWS Region where your Amazon S3 resources are located.
5. Choose **Create**.
6. The newly created connector will appear in the **Connectors** list.

Connect to Amazon Aurora

To connect App Studio with Aurora to enable builders to access and use Aurora resources in applications, you must perform the following steps:

1. [Create and configure Aurora resources](#)
2. [Create an IAM role to give App Studio access to Aurora resources](#)
3. [Create Aurora connector](#)

Create and configure Aurora resources

Creating an Aurora PostgreSQL-Compatible cluster

Use the following procedure to create and configure Aurora functions to be used with App Studio.

To set up Aurora for use with App Studio

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Create database**.
3. Choose **Aurora (PostgreSQL Compatible)**.
4. In **Available versions**, choose any version greater than or equal to version 13.11, 14.8, and 15.3.
5. In **Settings**, enter a **DB cluster identifier**.
6. In **Instance configuration**, choose **Serverless v2** and choose an appropriate capacity.
7. In **Connectivity**, select **Enable the RDS Data API**.
8. In **Database authentication**, select **IAM database authentication**.
9. In **Additional configuration**, in **Initial database name**, enter an initial database name for your database.

Create an IAM role to give App Studio access to Aurora resources

To use Aurora resources with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of data that builders can use and what operations can be called against that data, such as Create, Read, Update, or Delete.

We recommend creating at least one IAM role per service and policy.

To create an IAM role to give App Studio access to Aurora resources

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/IsAppStudioAccessRole": "true"
        }
      }
    }
  ]
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the + next to a policy will expand the policy to show the permissions granted by it. For Aurora, you may consider adding the AmazonRDSDataFullAccess policy, which grants permissions to invoke Aurora functions.

For more information about using IAM policies with Aurora, including a list of managed policies and their descriptions, see [Identity and Access Management for Amazon Aurora](#) in the *AWS Lambda Developer Guide*.

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key:** `IsAppStudioDataAccessRole`
 - **Value:** `true`
8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it when [creating the Aurora connector in App Studio](#).

Create Aurora connector

To create a connector for Aurora

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **AWS Services** from the list of connector types.
4. Configure your connector by filling out the following fields:
 - **Name:** Enter a name for your Aurora connector.
 - **Description:** Enter a description for your Aurora connector.
 - **IAM role:** Enter the Amazon Resource Name (ARN) from the IAM role created in [Create an IAM role to give App Studio access to Aurora resources](#). For more information about IAM, see the [IAM User Guide](#).
 - **Service:** Choose **Aurora**.
 - **Region:** Choose the AWS Region where your Aurora resources are located.
5. Choose **Create**.
6. The newly created connector will appear in the **Connectors** list.

Connect to Amazon Bedrock

To connect App Studio with Amazon Bedrock so builders can access and use Amazon Bedrock in applications, you must perform the following steps:

1. [Enable Amazon Bedrock models](#)
2. [Create an IAM role to give App Studio access to Amazon Bedrock](#)
3. [Create Amazon Bedrock connector](#)

Enable Amazon Bedrock models

Use the following procedure to enable Amazon Bedrock models.

To enable Amazon Bedrock models

1. Sign in to the AWS Management Console and open the Amazon Bedrock console at <https://console.aws.amazon.com/bedrock/>.
2. In the left navigation pane, choose **Model access**.
3. Enable the models that you want to use. For more information, see [Manage access to Amazon Bedrock foundation models](#).

Create an IAM role to give App Studio access to Amazon Bedrock

To use Amazon Bedrock with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of permissions for App Studio apps to use, and is used when creating the connector. We recommend creating at least one IAM role per service and policy.

To create an IAM role to give App Studio access to Amazon Bedrock

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/IsAppStudioAccessRole": "true"
        }
      }
    }
  ]
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the **+** next to a policy will expand the policy to show the permissions granted by it. For Amazon Bedrock, you may consider adding the `AmazonBedrockFullAccess` policy, which grants full access to Amazon Bedrock.

For more information about using IAM policies with Amazon Bedrock, including a list of managed policies and their descriptions, see [Identity and Access Management for Amazon Bedrock](#) in the *Amazon Bedrock User Guide*.

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key:** `IsAppStudioDataAccessRole`
 - **Value:** `true`

8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it when creating the Amazon Bedrock connector in App Studio in the next step.

Create Amazon Bedrock connector

To create a connector for Amazon Bedrock

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **AWS Services** from the list of connector types.
4. Configure your connector by filling out the following fields:
 - **Name:** Enter a name for your Amazon Bedrock connector.
 - **Description:** Enter a description for your Amazon Bedrock connector.
 - **IAM role:** Enter the Amazon Resource Name (ARN) from the IAM role created in [Create an IAM role to give App Studio access to Amazon Bedrock](#). For more information about IAM, see the [IAM User Guide](#).
 - **Service:** Choose **Bedrock Runtime**.
 - **Region:** Choose the AWS Region where your Amazon Bedrock resources are located.
5. Choose **Create**.
6. The newly created connector will appear in the **Connectors** list.

Connect to AWS services using the Other AWS services connector

While App Studio offers some connectors that are specific to certain AWS services, you can also connect to other AWS services using the **Other AWS services** connector.

Note

It is recommended to use the connector specific to the AWS service if it is available. For a list of available connectors for AWS services, see [Connect to AWS services](#).

To connect App Studio with AWS services to enable builders to access and use the service's resources in applications, you must perform the following steps:

1. [Create an IAM role to give App Studio access to AWS resources](#)
2. [Create an Other AWS services connector](#)

Create an IAM role to give App Studio access to AWS resources

To use AWS services and resources with App Studio, administrators must create an IAM role to give App Studio permissions to access the resources. The IAM role controls the scope of resources that builders can access and what operations can be called against the resources. We recommend creating at least one IAM role per service and policy.

To create an IAM role to give App Studio access to AWS resources

1. Sign in to the [IAM console](#) with a user that has permissions to create IAM roles. We recommend using the administrative user created in [Create an administrative user for managing AWS resources](#).
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. In **Trusted entity type**, choose **Custom trust policy**.
4. Replace the default policy with the following policy to allow App Studio applications to assume this role in your account.

You must replace **111122223333** with the AWS account number of the account used to set up the App Studio instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/IsAppStudioAccessRole": "true"
        }
      }
    }
  ]
}
```



```
}
```

Choose **Next**.

5. In **Add permissions**, search and choose the policies that grant the appropriate permissions for the role. Choosing the **+** next to a policy will expand the policy to show the permissions granted by it. For more information about IAM, see the [IAM User Guide](#).

Choose **Next**.

6. In **Role details**, provide a name and description.
7. In **Step 3: Add tags**, choose **Add new tag** to add the following tag to provide App Studio access:
 - **Key:** IsAppStudioDataAccessRole
 - **Value:** true
8. Choose **Create role** and make note of the generated Amazon Resource Name (ARN), you will need it when [creating the Other AWS services connector in App Studio](#).

Create an Other AWS services connector

To connect to AWS services using the Other AWS services connector

1. In the navigation pane, choose **Connectors** in the **Manage** section.
2. Choose **+ Create connector**.
3. Choose **Other AWS services** in the **AWS connectors** section of the supported services list.
4. Configure your AWS service connector by filling out the following fields:
 - **Name:** Provide a name for your connector.
 - **Description:** Provide a description for your connector.
 - **IAM role:** Enter the Amazon Resource Name (ARN) from the IAM role that was created in [Create an IAM role to give App Studio access to AWS resources](#).
 - **Service:** Select the AWS service you want to connect to App Studio.
 - **Region:** Select the AWS Region where your AWS resources are located.
5. Choose **Create**. The newly created connector will appear in the connectors list.

Connect to third-party services

Topics

- [Connect to third-party services and APIs \(generic\)](#)
- [Connect to services with OpenAPI](#)
- [Connect to Salesforce](#)

Connect to third-party services and APIs (generic)

Use the following procedure to create a generic **API Connector** in App Studio. The **API Connector** is used to provide App Studio apps with access to third-party services, resources, or operations.

To connect to third-party services with the API Connector

1. In the navigation pane, choose **connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **API Connector**. Now, configure your connector by filling out the following fields.
4. **Connector name:** Provide a name for your connector.
5. **Connector description:** Provide a description for your connector.
6. **Base URL:** The website or host of the third-party connection. For example, `www.slack.com`.
7. **Authentication method:** Choose the method for authenticating with the target service.
 - **None:** Access the target service with no authentication.
 - **Basic:** Access the target service using a **Username** and **Password** obtained from the service being connected to.
 - **Bearer Token:** Access the target service using the **Token value** of an authentication token obtained from the service's user account or API settings.
 - **OAuth 2.0:** Access the target service using the OAuth 2.0 protocol, which grants App Studio access to the service and resources without sharing any credentials or identity. To use the OAuth 2.0 authentication method, you must first create an application from the service being connected to that represents App Studio to obtain the necessary information. With that information, fill out the following fields:
 - a. **Client credentials flow:** Ideal for system-to-system interactions where the application acts on its own behalf without user interaction. For example, a CRM app that updates

Salesforce records automatically based on new records added by users, or an app that retrieves and displays transaction data in reports.

1. In **Client ID**, enter the ID obtained from the OAuth app created in the target service.
2. In **Client secret**, enter the secret obtained from the OAuth app created in the target service.
3. In **Access token URL**, enter the token URL obtained from the OAuth app created in the target service.
4. Optionally, in **Scopes**, enter the scopes for the application. Scopes are permissions or access levels required by the application. Refer to the target service's API documentation to understand their scopes, and configure only those that your App Studio app needs.

Choose **Verify connection** to test the authentication and connection.

- b. **Authorization code flow:** Ideal for applications that require acting on behalf of a user. For example, a customer support app where users log in and view and update support tickets, or a sales app where each team members logs in to view and manage their sales data.
 1. In **Client ID**, enter the ID obtained from the OAuth app created in the target service.
 2. In **Client secret**, enter the secret obtained from the OAuth app created in the target service.
 3. In **Authorization URL**, enter the authorization URL from the target service.
 4. In **Access token URL**, enter the token URL obtained from the OAuth app created in the target service.
 5. Optionally, in **Scopes**, enter the scopes for the application. Scopes are permissions or access levels required by the application. Refer to the target service's API documentation to understand their scopes, and configure only those that your App Studio app needs.
8. **Headers:** Add HTTP headers that are used to provide metadata about the request or response. You can add both keys and values, or only provide a key to which the builder can provide a value in the application.

9. **Query parameters:** Add query parameters that are used to pass options, filters, or data as part of the request URL. Like headers, you can provide both a key and value, or only provide a key to which the builder can provide a value in the application.
10. Choose **Create**. The newly created connector will appear in the **Connectors** list.

Now that the connector is created, builders can use it in their apps.

Connect to services with OpenAPI

To connect App Studio with services using OpenAPI to enable builders to build applications that send requests and receive responses from the services, perform the following steps:

1. [Get the OpenAPI Specification file and gather service information](#)
2. [Create OpenAPI connector](#)

Get the OpenAPI Specification file and gather service information

To connect a service to App Studio with OpenAPI, perform the following steps:

1. Go to the service that you want to connect to App Studio and find an OpenAPI Specification JSON file.

Note

App Studio supports OpenAPI Specification files that conform to version OpenAPI Specification Version 3.0.0 or higher.

2. Gather the necessary data to configure the OpenAPI connector, including the following:
 - The base URL for connecting to the service.
 - Authentication credentials, such as a token or username/password.
 - If applicable, any headers.
 - If applicable, any query parameters.

Create OpenAPI connector

To create a connector for OpenAPI

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **OpenAPI Connector** from the list of connector types. Now, configure your connector by filling out the following fields.
4. **Name:** Enter a name for your OpenAPI connector.
5. **Description:** Enter a description for your OpenAPI connector.
6. **Base URL:** Enter the base URL for connecting to the service.
7. **Authentication method:** Choose the method for authenticating with the target service.
 - **None:** Access the target service with no authentication.
 - **Basic:** Access the target service using a **Username** and **Password** obtained from the service being connected to.
 - **Bearer Token:** Access the target service using the **Token value** of an authentication token obtained from the service's user account or API settings.
 - **OAuth 2.0:** Access the target service using the OAuth 2.0 protocol, which grants App Studio access to the service and resources without sharing any credentials or identity. To use the OAuth 2.0 authentication method, you must first create an application from the service being connected to that represents App Studio to obtain the necessary information. With that information, fill out the following fields:
 - a. **Client credentials flow:**
 1. In **Client ID**, enter the ID from the target service.
 2. In **Client secret**, enter the secret from the target service.
 3. In **Access token URL**, enter the token URL from the target service.
 4. Optionally, in **Scopes**, enter the scopes for the application. Scopes are permissions or access levels required by the application. Refer to the target service's API documentation to understand their scopes, and configure only those that your App Studio app needs.

Add any **Variables** to be sent with the service with each call, and choose **Verify connection** to test the authentication and connection.

b. **Authorization code flow:**

1. In **Client ID**, enter the ID from the target service.
 2. In **Client secret**, enter the secret from the target service.
 3. In **Authorization URL**, enter the authorization URL from the target service.
 4. In **Access token URL**, enter the token URL from the target service.
 5. Optionally, in **Scopes**, enter the scopes for the application. Scopes are permissions or access levels required by the application. Refer to the target service's API documentation to understand their scopes, and configure only those that your App Studio app needs.
8. **Variables:** Add variables to be sent to the service with each call. Variables added during configuration are securely stored and only accessed during runtime of applications that use the connection.
 9. **Headers:** Add HTTP headers that are used to provide metadata about the request or response. You can add both keys and values, or only provide a key to which the builder can provide a value in the application.
 10. **Query parameters:** Add query parameters that are used to pass options, filters, or data as part of the request URL. Like headers, you can provide both a key and value, or only provide a key to which the builder can provide a value in the application.
 11. **OpenAPI Spec File:** Upload an OpenAPI Specification JSON file by dragging and dropping, or choosing **Select a file** to navigate your local file system and choose the file to be uploaded.

Once added, the file is processed and a list of available options are displayed. Select the necessary operations for your connector.

12. Choose **Create**. The newly created connector will appear in the **Connectors** list.

Now that the connector is created, builders can use it in their apps.

Connect to Salesforce

To connect App Studio with Salesforce to enable builders to access and use Salesforce resources in applications, you must create and configure a connected app in Salesforce and create a Salesforce connector in App Studio.

To connect Salesforce with App Studio

1. In App Studio, in the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with some details about each.
2. Choose **+ Create connector**.
3. Choose **Salesforce** from the list of connector types to open the connector creation page.
4. Take note of the **Redirect URL**, which you will use to configure Salesforce in the following steps.
5. The next step is to create a connected app in Salesforce. In another tab or window, navigate to your Salesforce instance.
6. In the Quick Find box, search **App Manager** and then select **App Manager**.
7. Choose **New Connected App**.
8. In **Connected App Name** and **API Name**, enter a name for your app. It does not have to match your App Studio app name.
9. Provide contact information as needed.
10. In the **API (Enable OAuth Settings)** section, enable **Enable OAuth Settings**.
11. In **Callback URL**, enter the **Redirect URL** you noted earlier from App Studio.
12. In **Selected OAuth Scopes**, add the necessary permissions scopes from the list. App Studio can interact with Salesforce REST APIs to perform CRUD operations on five objects: Accounts, Cases, Contacts, Leads, and Opportunities. It is recommended to add **Full access (full)** to ensure that your App Studio app has all relevant permissions or scopes.
13. Enable **Require Secret for Web Server Flow** and **Require Secret for Refresh Token Flow** to follow the best security practices.
14. App Studio supports both of the following authentication flows:
 - **Client Credentials Flow**: Ideal for server-to-server interactions where the application acts on its own behalf without user interaction. For example, listing all leads information for a team of temporary employees who do not have Salesforce access.

- **Authorization Code Flow:** Suitable for applications that act on behalf of a user, such as personal data access or actions. For example, listing each sales manager's leads sourced or owned by them to perform other tasks through this app.
 - For the Client Credentials Flow:
 - a. Enable **Enable Client Credentials Flow**. Review and confirm the message.
 - b. Save the app.
 - c. You must select an execution user, although there is no user interaction in the flow. By selecting an execution user, Salesforce returns access tokens on behalf of the user.
 1. In the **App Manager**, from the list of apps, choose the arrow of the App Studio app and choose **Manage**.
 2. Choose **Edit Policies**
 3. In **Client Credentials Flow**, add the appropriate user.
 - For the Authorization Code Flow, enable **Enable Authorization Code and Credentials Flow**
15. Salesforce provides a Client ID and Client Secret, which must be used to configure the connector in App Studio in the following steps.
- a. In the **App Manager**, choose the arrow of the App Studio app and choose **View**.
 - b. In the **API (Enable OAuth Settings)** section, choose **Manage Consumer Details** . This may send an email for a verification key, which you need to enter for confirmation.
 - c. Note the **Consumer Key** (Client ID) and the **Consumer Secret** (Client Secret).
16. Back in App Studio, configure and create your connector by filling out the following fields.
17. In **Name**, enter a name for your Salesforce connector.
18. In **Description**, enter a description for your Salesforce connector.
19. In **Base URL**, enter the base URL for your Salesforce instance. It should look like this:
`https://hostname.salesforce.com/api/services/data/v60.0`, replacing *hostname* with your Salesforce instance name.
20. In **Authentication method**, ensure **OAuth 2.0** is selected.
21. In **OAuth 2.0 Flow**, select the OAuth authentication method and fill out the related fields:
- Select **Client credentials flow** for use in applications that act on their own behalf, for system-to-system integrations.

- a. In **Client ID**, enter the **Consumer Key** obtained previously from Salesforce.
 - b. In **Client secret**, enter the **Consumer Secret**, obtained previously from Salesforce.
 - c. In **Access token URL**, enter the OAuth 2.0 token endpoint. It should look like this: `https://hostname/services/oauth2/token`, replacing *hostname* with your Salesforce instance name. For more information, see the [Salesforce OAuth Endpoints](#) documentation.
 - d. Choose **Verify connection** to test the authentication and connection.
- Select **Authorization code flow** for use in applications that act on behalf of the user.
 - a. In **Client ID**, enter the **Consumer Key** obtained previously from Salesforce.
 - b. In **Client secret**, enter the **Consumer Secret**, obtained previously from Salesforce.
 - c. In **Authorization URL**, enter the authorization endpoint. It should look like this: `https://hostname/services/oauth2/authorize`, replacing *hostname* with your Salesforce instance name. For more information, see the [Salesforce OAuth Endpoints](#) documentation.
 - d. In **Access token URL**, enter the OAuth 2.0 token endpoint. It should look like this: `https://hostname/services/oauth2/token`, replacing *hostname* with your Salesforce instance name. For more information, see the [Salesforce OAuth Endpoints](#) documentation.
22. In **Operations**, select the Salesforce operations that your connector will support. The operations in this list are predefined and represent common tasks within Salesforce, such as creating, retrieving, updating, or deleting records from common objects.
 23. Choose **Create**. The newly created connector will appear in the **Connectors** list.

Viewing, editing, and deleting connectors

To view, edit, or delete existing connectors

1. In the navigation pane, choose **Connectors** in the **Manage** section. You will be taken to a page displaying a list of existing connectors with the following details for each connector:
 - **Name:** The name of the connector that was provided during creation.
 - **Description:** The description of the connector that was provided during creation.

- **Connected to:** The service that the connector is connecting to App Studio. A value of **API** represents a connection to a third-party service.
 - **Created by:** The user that created the connector.
 - **Date created:** The date that the connector was created.
2. To view more details about a connector, or edit or delete a connector, use the following instructions:
- To see more information about a specific connector, choose **View** for that connector.
 - To edit a connector, choose the dropdown menu next to **View** and choose **Edit**.
 - To delete a connector, choose the dropdown menu next to **View** and choose **Delete**.

Builder documentation

The following topics contain information to help users in App Studio who are creating, editing, and publishing applications.

Topics

- [Creating, editing, and deleting applications](#)
- [Previewing, publishing, and sharing applications](#)
- [Building your app's user interface with pages and components](#)
- [Defining and implementing your app's business logic with automations](#)
- [Configure your app's data model with entities](#)
- [Page and automation parameters](#)
- [Generative AI in App Studio](#)
- [Using JavaScript to write expressions in App Studio](#)
- [Troubleshooting and debugging App Studio apps](#)
- [Building an app with multiple users](#)

Creating, editing, and deleting applications

Contents

- [Viewing applications](#)
- [Creating an application](#)
- [Editing an application](#)
 - [Application settings](#)
 - [App navigation](#)
- [Deleting an application](#)

Viewing applications

Use the following procedure to view applications in App Studio.

To view applications

1. In the navigation pane, choose **My applications** in the **Build** section. You will be taken to a page displaying a list of applications that you have access to.
2. On the **My applications** page, a table displays a list of your applications with the following details:
 - **Application name:** The name of the application.
 - **Status:** The status of the application. The possible values are:
 - **Draft:** The application has not been published.
 - **Published:** The application has been published.
 - **Last updated:** The date that the application was last edited.
 - **Role:** Your role in relation to the application. The possible values are:
 - **Owner:** App **owners** have all access and permissions to the app.
 - **Co-owner:** App **co-owners** have access similar to app **owners**.
 - **Edit-only:** Users with **Edit-only** access to an app can edit the app, but cannot invite other builders to the app, publish the app to production, add new data sources, delete the app, or clone the app.
3. You can choose the arrow in the **Actions** column to open the actions menu for that application with the following options:
 - **Edit:** Opens the app for editing in the builder studio. Editing is only available to app owners and editors.
 - **Share:** Opens a dialog box where the app link can be copied. Sharing is only available on published applications.
 - **View:** Opens the running application. Viewing is only available on published applications.
 - **Duplicate:** Create another app with the same components, automations, and entities as the current app.
 - **Rename:** Provide a new name for the app.
 - **Delete:** Deletes the application. Deletion is only available to app owners and admins.

Creating an application

Use the following procedure to create an application in App Studio.

To create an application

1. There are two options for creating an application in App Studio:
 - In the navigation pane, choose **Builder hub**. Choose **Get started** in the banner.
 - In the navigation pane, choose **My applications** in the **Build** section. You will be taken to a page displaying a list of applications that you have access to.

Choose **+ Create app**.

2. In the **Create app** dialog box, give your application a name.
3. Choose **Create** to create your application. You will be taken to the application studio where you can use components, automations, and data to configure the look and function of your application. For information about building applications, see [Getting started with AWS App Studio](#).

Editing an application

Use the following procedure to edit an application in App Studio.

To edit an application

1. In the navigation pane, choose **My applications** in the **Build** section. You will be taken to a page displaying a list of applications that you have access to.
2. Choose the dropdown in the **Actions** column of the application you want to edit.
3. To rename an application, choose **Rename**, give your application a new name, and choose **Rename**.
4. To edit an application, choose **Edit**. This will take you to the application studio where you can use components, automations, and data to configure the look and function of your application. For information about building applications, see [Getting started with AWS App Studio](#).

Application settings

In the application studio, you can view and update the following application settings.

App navigation

By default, App Studio shows all pages in the app navigation of published apps or when previewing apps. You can reorder the pages or remove pages from the navigation in the **App navigation** section, which contains the following settings:

- The **Show navigation for these pages** toggle defines if app users can navigate to the defined pages in your app.
- In **Home page**, choose the page you want app users to be navigated to when first accessing your app from the dropdown.
- In **Other pages**, choose if pages can be navigated to, and which order they are displayed in the app navigation menu.

Deleting an application

Use the following procedure to delete an application in App Studio.

To delete an application

1. In the navigation pane, choose **My applications** in the **Build** section. You will be taken to a page displaying a list of applications that you have access to.
2. Choose the dropdown in the **Actions** column of the application you want to delete.
3. Choose **Delete**.
4. In the **Delete application** dialog box, carefully review the information about deleting applications. If you want to delete the application, choose **Delete**.

Previewing, publishing, and sharing applications

Topics

- [Previewing applications](#)
- [Publishing applications](#)
- [Sharing published applications](#)

Previewing applications

You can preview applications in App Studio to see how they will appear to users and also test its functionality by using it and checking logs in a debug panel.

The application preview environment does not support displaying live data or the connection with external resources with connectors, such as data sources. To test functionality in the preview environment, you can use mocked output in automations and sample data in entities. To view your application with real-time data, you must publish your app. For more information, see [Publishing applications](#).

The preview or development environment does not update the application published in the other environments. If an application has not been published, users will not be able to access it until it is published and shared. If an application has already been published and shared, users will still access the version that has been published, and not the version used in a preview environment.

To preview your application

1. If necessary, navigate to the application studio of the application you want to preview:
 - a. In the navigation pane, choose **My applications** in the **Build** section.
 - b. Choose **Edit** for the application.
2. Choose **Preview** to open the preview environment for the application.
3. (Optional) Expand the debug panel by choosing its header near the bottom of the screen. You can filter the panel by type of message by choosing the type of message in the **Filter logs** section. You can clear the panel's logs by choosing **Clear console**.
4. While in the preview environment, you can test your application by navigating around its pages, using its components, and choosing its buttons to start automations that transfer data. Because the preview environment doesn't support live data or connections to external sources, you can view examples of the data being transferred in the debug panel.

Publishing applications

When you've finished creating and configuring your application the next step is to publish it to test data transfers or share it with end users. To understand publishing applications in App Studio, it's important to understand the available environments. App Studio provides three separate environments, which are described in the following list:

1. **Development:** Where you build and preview your application. You do not need to publish to the Development environment, as the latest version of your application is hosted there automatically. No live data or third-party services or resources are available in this environment.
2. **Testing:** Where you can perform comprehensive testing of your application. In the Testing environment, you can connect to, send data to, and receive data from other services.
3. **Production:** The live operational environment for end-user consumption.

All of your app building takes place in the **Development** environment. Then, publish to the **Testing** environment to test data transfer between other services, and user acceptance testing (UAT) by providing an access URL to end users. Afterwards, publish your app to the **Production** environment to perform final tests before sharing it with users. For more information about the application environments, see [Application environments](#).

When you publish an application, it is not available for users until it is shared. This gives you the opportunity to use and test the application in the Testing and Production environments before users can access it. When you publish an application to Production that has previously been published and shared, the version that is available to users is updated.


Publishing applications

Use the following procedure to publish an App Studio application to the Testing or Production environment.

To publish an application to Testing or Production environment

1. In the navigation pane, choose **My applications** in the **Build** section. You will be taken to a page displaying a list of applications that you have access to.
2. Choose **Edit** for the application you want to publish.
3. Choose **Publish** in the top-right corner.
4. In the **Publish your updates** dialog box:
 - a. Review the information about publishing an application.
 - b. (Optional) In **Version description**, include a description of this version of the application.
 - c. Choose the box to acknowledge the information about the environment.
 - d. Choose **Start**. It can take up to 15 minutes for the application to be updated in the live environment.

5. For information about viewing applications in the Testing or Production environments, see [Viewing published applications](#).

 **Note**

Using the application in the Testing or Production environment will result in live data transfer, such as creating records in tables of data sources that have been connected with connectors.


Published applications that have never been shared will not be available to users or other builders. To make an application available to users, you must share it after publishing. For more information, see [Sharing published applications](#).

Viewing published applications

You can view applications published to the Testing and Production environments to test the application before sharing it with end users or other builders.

To view published applications in the Testing or Production environment

1. If necessary, navigate to the application studio of the application you want to preview:
 - a. In the navigation pane, choose **My applications** in the **Build** section.
 - b. Choose **Edit** for the application.
2. Choose the dropdown arrow next to **Publish** in the top-right corner and choose **Publish Center**.
3. From the publishing center, you can view the environments that your application is published to. If your application is published to the Testing or Production environments, you can view the app using the **URL** link for each environment.

 **Note**

Using the application in the Testing or Production environment will result in live data transfer, such as creating records in tables of data sources that have been connected with connectors.

Application environments

AWS App Studio provides application lifecycle management (ALM) capabilities with three separate environments - Development, Testing, and Production. This helps you more easily best practices such as maintaining separate environments, version control, sharing, and monitoring across the entire app lifecycle.

Development environment

The **Development** environment is an isolated sandbox where you can build apps without connecting to any live data sources or services using the application studio and sample data. In the Development environment, you can preview your app to view and test the app without compromising production data.

Although your app doesn't connect to other services in the Development environment, you can configure different resources in your app to mimic live data connectors and automations.

There is a collapsible debug panel that includes errors and warnings at the bottom of the application studio in the Development environment to help you inspect and debug the app as you build. For more information about troubleshooting and debugging apps, see [Troubleshooting and debugging App Studio apps](#).

Testing environment

Once your initial app development is complete, the next step is to publish to the **Testing** environment. While in the Testing environment, your app can connect to, send data to, and receive data from other services. Therefore, you can use this environment to perform comprehensive testing including user acceptance testing (UAT) by providing an access URL to end users.

Note

Your initial publish to the Testing environment may take up to 15 minutes.

The version of your app published to the Testing environment will be removed after 3 hours of end-user inactivity. However, all versions persist and can be restored from the **Version History** tab.

Key features of the Testing environment are as follows:

- Integration testing with live data sources and APIs
- User acceptance testing (UAT) facilitated through controlled access

- Environment for gathering feedback and addressing issues
- Ability to inspect and debug both client-side and server-side activities using browser consoles and developer tools.

For more information about troubleshooting and debugging apps, see [Troubleshooting and debugging App Studio apps](#).

Production environment

After you have tested and fixed any issues, you can promote the version of your application from the Testing environment to the Production environment for live operational use. Although the Production environment is the live operational environment for end-user consumption, you can test the published version before sharing it with users.

Your published version in the Production environment will be removed after 14 days of end-user inactivity. However, all versions persist and can be restored from the **Version History** tab.

Key features of the Production environment are as follows:

- Live operational environment for end-user consumption
- Granular role-based access control
- Version control and rollback capabilities
- Ability to inspect and debug client-side activities only
- Uses live connectors, data, automations, and APIs

Versioning and release management

App Studio provides version control and release management capabilities through its versioning system in the **Publish center**.

Key versioning capabilities:

- Publishing to the Testing environment generates new version numbers (1.0, 2.0, 3.0...).
- The version number does not change when promoting from the Testing to Production environment.
- You can roll back to any previous version from **Version History**.
- Applications published to the Testing environment are paused after 3 hours of inactivity. Versions are persisted and can be restored from **Version History**.

- Applications published to the Production environment are removed after 14 days of inactivity. Versions are persisted and can be restored from **Version History**.

This versioning model allows for rapid iteration while maintaining traceability, rollback capabilities, and optimal performance across the app development and testing cycle.

Maintenance and operations

App Studio may need to automatically republish your application to address certain maintenance tasks, operational activities, and to incorporate new software libraries. No action is needed from you, the builder, but end users may need to log back into the application. In certain situations, we may need you to republish your application to incorporate new features and libraries which we cannot automatically add ourselves. You will need to resolve any errors and review warnings before republishing.

Sharing published applications

When you publish an application that has not been published yet, it is not available for users until it is shared. Once a published application has been shared, it will be available to users and will not need to be shared again if another version is published.

Note

This section is about sharing published applications with end users or testers. For information about inviting other users to build an app, see [Building an app with multiple users](#).

To share a published application

1. Access the **Share** dialog box from either the application list, or the application studio of your app by using the following instructions:
 - To access the **Share** dialog box from the application list: In the navigation pane, choose **My applications** in the **Build** section. Choose the dropdown in the **Actions** column of the application you want to share and choose **Share**.
 - To access the **Share** dialog box from the application studio: From the application studio of your app, choose **Share** in the top header.

2. In the **Share** dialog box, choose the tab for the environment that you want to share. If you do not see the **Testing** or **Production** tabs, your app may not be published to the corresponding environment. For more information about publishing, see [Publishing applications](#).
3. In the appropriate tab, select groups from the dropdown menu to share the environment with them.
4. (Optional) Assign an app-level role to the group for testing or configuring conditional page visibility. For more information, see [Configuring role-based visibility of pages](#).
5. Choose **Share**.
6. (Optional) Copy and share the link with users. Only users that the application and environment have been shared with can access the application in the corresponding environment.

Building your app's user interface with pages and components

Topics

- [Creating, editing, or deleting pages](#)
- [Adding, editing, and deleting components](#)
- [Configuring role-based visibility of pages](#)
- [Components reference](#)

Creating, editing, or deleting pages

Use the following procedures to create, edit, or delete pages from your AWS App Studio application.

Pages are containers for [components](#), which make up the UI of an application in App Studio. Each page represents a screen of your application's user interface (UI) that your users will interact with. Pages are created and edited in the **Pages** tab of the application studio.

Creating a page

Use the following procedure to create a page in an application in App Studio.

To create a page

1. If necessary, navigate to the application studio of your application.

2. Navigate to the **Pages** tab of the application studio.
3. In the left-side **Pages** menu, choose **+ Add**.

Viewing and editing page properties

Use the following procedure to edit a page in an application in App Studio. You can edit properties such as the page's name, its parameters, and its layout.

To view or edit page properties

1. If necessary, navigate to the application studio of your application.
2. Navigate to the **Pages** tab of the application studio.
3. In the left-side **Pages** menu, choose the ellipses menu next to the name of the page you want to edit and choose **Page properties**. This opens the right-side **Properties** menu.
4. To edit the page name:

Note

Valid page name characters: **A-Z, a-z, 0-9, _, \$**

- a. Choose the pencil icon next to the name near the top of the **Properties** menu.
 - b. Enter the new name for your page and press Enter.
5. To create, edit, or delete page parameters:
 - a. To create a page parameter, choose **+ Add new** in the **Page parameters** section.
 - b. To edit a page parameter's **Key** or **Description** value, choose input field of the property you want to change and enter a new value. Your changes are saved as you edit.
 - c. To delete a page parameter, choose the trash icon of the page parameter you want to delete.
 6. To add, edit, or remove a page's logo or banner:
 - a. To add a page logo or banner, enable the respective option in the **Style** section. Configure the image's source and optionally provide alt text.
 - b. To edit a page logo or banner, update the fields in the **Style** section.
 - c. To remove a page logo or banner, disable the respective option in the **Style** section.

7. To edit a page's layout:
 - Update the fields in the **Layout** section.

Deleting a page

Use the following procedure to delete a page from an application in App Studio.

To delete a page

1. If necessary, navigate to the application studio of your application.
2. Navigate to the **Pages** tab of the application studio.
3. In the left-side **Pages** menu, choose the ellipses menu next to the name of the page you want to delete and choose **Delete**.

Adding, editing, and deleting components

Use the following procedures to add, edit, and delete components in or from pages in the App Studio application studio to craft the desired user interface for your application.

Adding components to a page

1. If necessary, navigate to the application studio of your application.
2. Navigate to the **Pages** tab of the application studio.
3. The components panel is located in the right-side menu, which contains the available components.
4. Drag and drop the desired component from the panel onto the canvas. Alternatively, you can double-click on the component in the panel to automatically add it to the center of the current page.
5. Now that you've added a component, use the right-side **Properties** panel to adjust its settings, such as the data source, layout, and behavior. For detailed information about configuring each component type, see [Components reference](#).

Viewing and editing component properties

1. If necessary, navigate to the application studio of your application.

2. Navigate to the **Pages** tab of the application studio.
3. In the left-side **Pages** menu, expand the page that contains the component and choose the component to be viewed or edited. Alternatively, you can choose the page and then choose the component from the canvas.
4. The right-side **Properties** panel displays the configurable settings for the selected component.
5. Explore the various properties and options available, and update them as necessary to configure the component's appearance and behavior. For example, you might want to change the data source, configure the layout, or enable additional functionality.

For detailed information about configuring each component type, see [Components reference](#).

Deleting components

1. If necessary, navigate to the application studio of your application.
2. Navigate to the **Pages** tab of the application studio.
3. In the left-side **Pages** menu, choose the component to be deleted to select it.
4. In the right-side **Properties** menu, choose the trash icon.
5. In the confirmation dialog box, choose **Delete**.

Configuring role-based visibility of pages

You can create roles within an App Studio app and configure the visibility of pages based on those roles. For example, you can create roles based on user needs or access levels, such as administrator, manager, or user for apps that provide features such as project approvals or claims processing and make certain pages visible to specific roles. In this example, administrators may have full access, managers might have access to view reporting dashboards, and users may have access to tasks pages with input forms.

Use the following procedure to configure role-based visibility of pages in your App Studio app.

1. If necessary, navigate to the application studio of your application. From the left-side navigation menu, choose **My applications**, find your application and choose **Edit**.
2. Create app level roles in the application studio.
 - a. Choose the **App settings** tab at the top of the application studio.

- b. Choose **+ Add Role**
 - c. In **Role name**, provide a name to identify your role. We recommend using a name that is descriptive of the group's access level or duties, as you'll use the name to set up the page visibility.
 - d. Optionally, in **Description**, add a description for the role.
 - e. Repeat these steps to create as many roles as needed.
3. Configure the visibility of your pages
 - a. Choose the **Pages** tab at the top of the application studio.
 - b. From the left-side **Pages** menu, choose the page for which you want to configure role-based visibility.
 - c. In the right-side menu, choose the **Properties** tab.
 - d. In **Visibility**, disable **Open to all end users**.
 - e. Keep **Role** selected to choose from a list of the roles you created in the previous step. Choose **Custom** to write a JavaScript expression for more complex visibility configurations.
 1. With **Role** selected, check the boxes of the app roles for which the page will be visible.
 2. With **Custom** selected, enter a JavaScript expression that resolves to true or false. Use the following example to check if the current user has the role of *manager*:

```
{{currentUser.roles.includes('manager')}}.
```
 4. Now that your visibility is configured, you can test the page visibility by previewing your app.
 - a. Choose **Preview** to open a preview of your app.
 - b. In the top right of the preview, choose the **Previewing as** menu and check the boxes of the roles you want to test. The visible pages should reflect the roles selected.
 5. Now, assign groups to app roles for a published app. Group and role assignments must be configured separately for each environment. For more information about app environments, see [Application environments](#).

Note

Your app must be published to either the Testing or Production environments to assign App Studio groups to the roles you've created and configured. If necessary, publish your app to assign groups to the roles. For more information about publishing, see [Publishing applications](#).

- a. In the top right of the application studio, choose **Share**.
- b. Choose the tab for the environment of which you want to configure page visibility.
- c. Choose the **Search groups** input box and choose the group with which to share the app version. You can enter text to search for groups.
- d. In the dropdown menu, choose the roles to assign to the group. You can choose **No role** to share the app version and not assign a role to the group. Only pages that are visible to all users will be visible to groups with no role.
- e. Choose **Share**. Repeat these steps to add as many group as needed.

Components reference

This topic details each of App Studio's components, their properties, and includes configuration examples.

Common component properties

This section outlines the general properties and features that are shared across multiple components in the application studio. The specific implementation details and use cases for each property type may vary depending on the component, but the general concept of these properties remains consistent across App Studio.

Name

A default name is generated for each component; however, you can edit to change to a unique name to each component. You will use this name to reference the component and its data from other components or expressions within the same page. Limitation: Do not include spaces in the component name; it can only have letters, numbers, underscores and dollar signs. Examples: `userNameInput`, `ordersTable`, `metricCard1`.

Primary value, Secondary value, and Value

Many components in the application studio provide fields for specifying values or expressions that determine the content or data displayed within the component. These fields are often labeled as `Primary value`, `Secondary value`, or simply `Value`, depending on the component type and purpose.

The **Primary** value field is typically used to define the main value, data point, or content that should be prominently displayed within the component.

The **Secondary** value field, when available, is used to display an additional or supporting value or information alongside the primary value.

The **Value** field allows you to specify the value or expression that should be displayed in the component.

These fields support both static text input and dynamic expressions. By using expressions, you can reference data from other components, data sources, or variables within your application, enabling dynamic and data-driven content display.

Syntax for expressions

The syntax for entering expressions in these fields follows a consistent pattern:

```
{{expression}}
```

Where *expression* is a valid expression that evaluates to the desired value or data you want to display.

Example: Static text

- Primary value: you can enter a static number or value directly, such as "123" or "\$1,999.99".
- Secondary value: you can enter a static text label, such as "Goal" or "Projected Revenue".
- Value: you can enter a static string, such as "since last month" or "Total Quantity".

Examples: Expressions

- Hello, `{{currentUser.firstName}}`: Displays a greeting with the first name of the currently logged-in user.
- `{{currentUser.role === 'Admin' ? 'Admin Dashboard' : 'User Dashboard'}}`: Conditionally displays a different dashboard title based on the user's role.
- Last login: `{{currentUser.lastLoginDate.toLocaleDateString()}}`: Displays the last login date of the current user in a readable format.
- Signed in as: `{{currentUser.email}}`: Displays the email address of the current user.

- `{{currentUser.isSubscribed ? 'Subscribed' : 'Not Subscribed'}}`: Displays the subscription status of the current user.
- `{{ui.componentName.data?.[0]?.fieldName}}`: Retrieves the value of the `fieldName` field from the first item in the data of the component with the ID `componentName`.
- `{{ui.componentName.value * 100}}`: Performs a calculation on the value of the component with the ID `componentName`.
- `{{ui.componentName.value + ' items'}}`: Concatenates the value of the component with the ID `componentName` and the string `' items'`.
- `{{ui.ordersTable.data?.[0]?.orderNumber}}`: Retrieves the order number from the first row of data in the `ordersTable` component.
- `{{ui.salesMetrics.data?.[0]?.totalRevenue * 1.15}}`: Calculates the projected revenue by increasing the total revenue from the first row of data in the `salesMetrics` component by 15%.
- `{{ui.customerProfile.data?.[0]?.firstName + ' ' + ui.customerProfile.data?.lastName}}`: Concatenates the first and last name from the data in the `customerProfile` component.
- `{{new Date(ui.orderDetails.data?.orderDate).toLocaleDateString()}}`: Formats the order date from the `orderDetails` component to a more readable date string.
- `{{ui.productList.data?.length}}`: Displays the total number of products in the data connected to the `productList` component.
- `{{ui.discountPercentage.value * ui.orderTotal.value}}`: Calculates the discount amount based on the discount percentage and the order total.
- `{{ui.cartItemCount.value + ' items in cart'}}`: Displays the number of items in the shopping cart, along with the label `items in cart`.

By using these expression fields, you can create dynamic and data-driven content within your application, allowing you to display information that is tailored to the user's context or the state of your application. This enables more personalized and interactive user experiences.

Label

The **Label** property allows you to specify a caption or title for the component. This label is typically displayed alongside or above the component, helping users understand its purpose.

You can use both static text and expressions to define the label.

Example: Static text

If you enter the text "First Name" in the Label field, the component will display "First Name" as its label.

Example: Expressions

Example: Retail store

The following example personalizes the label for each user, making the interface feel more tailored to the individual:

```
{{currentUser.firstName}} {{currentUser.lastName}}'s Account
```

Example: SaaS project management

The following example pulls data from the selected project to provide context-specific labels, helping users stay oriented within the application:

```
Project {{ui.projectsTable.selectedRow.id}} - {{ui.projectsTable.selectedRow.name}}
```

Example: Healthcare clinic

The following example references the current user's profile and the doctor's information, providing a more personalized experience for patients.

```
Dr. {{ui.doctorProfileTable.data.firstName}}  
    {{ui.doctorProfileTable.data.lastName}}
```

Placeholder

The Placeholder property allows you to specify hint or guidance text that is displayed within the component when it is empty. This can help users understand the expected input format or provide additional context.

You can use both static text and expressions to define the placeholder.

Example: Static text

If you enter the text `Enter your name` in the **Placeholder** field, the component will display `Enter your name` as the placeholder text.

Example: Expressions

Example: Financial services

Enter the amount you'd like to deposit into your `{{ui.accountsTable.selectedRow.balance}}` account. These examples pull data from the selected account to display relevant prompts, making the interface intuitive for banking customers.

Example: E-commerce

Enter the coupon code for `{{ui.cartTable.data.currency}}` total. The placeholder here dynamically updates based on the user's cart contents, providing a seamless checkout experience.

Example: Healthcare clinic

Enter your `{{ui.patientProfile.data.age}}`-year-old patient's symptoms. By using an expression that references the patient's age, the application can create a more personalized and helpful placeholder.

Source

The **Source** property allows you to select the data source for a component. Upon selection, you can choose from the following data source types: `entity`, `expression`, or `automation`.

Entity

Selecting **Entity** as the data source allows you to connect the component to an existing data entity or model in your application. This is useful when you have a well-defined data structure or schema that you want to leverage throughout your application.

When to use the entity data source:

- When you have a data model or entity that contains the information you want to display in the component (e.g., a "Products" entity with fields like "Name", "Description", "Price").
- When you need to dynamically fetch data from a database, API, or other external data source and present it in the component.
- When you want to take advantage of the relationships and associations defined in your application's data model.

Selecting a query on an entity

Sometimes, you may want to connect a component to a specific query that retrieves data from an entity, rather than the entire entity. In the Entity data source, you have the option to choose from existing queries or create a new one.

By selecting a query, you can:

- Filter the data displayed in the component based on specific criteria.
- Pass parameters to the query to dynamically filter or sort the data.
- Leverage complex joins, aggregations, or other data manipulation techniques defined in the query.

For example, if you have a `Customers` entity in your application with fields like `Name`, `Email`, and `PhoneNumber`. You can connect a table component to this entity and choose a pre-defined `ActiveCustomers` data action that filters the customers based on their status. This allows you to display only the active customers in the table, rather than the entire customer database.

Adding parameters to an entity data source

When using an entity as the data source, you can also add parameters to the component. These parameters can be used to filter, sort, or transform the data displayed in the component.

For example, if you have a `Products` entity with fields like `Name`, `Description`, `Price`, and `Category`. You can add a parameter named `category` to a table component that displays the product list. When users select a category from a dropdown, the table will automatically update to show only the products belonging to the selected category, using the `{{params.category}}` expression in the data action.

Expression

Select **Expression** as the data source to enter custom expressions or calculations to generate the data for the component dynamically. This is useful when you need to perform transformations, combine data from multiple sources, or generate data based on specific business logic.

When to use the **Expression** data source:

- When you need to calculate or derive data that is not directly available in your data model (e.g., calculating the total order value based on quantity and price).

- When you want to combine data from multiple entities or data sources to create a composite view (e.g., displaying a customer's order history along with their contact information).
- When you need to generate data based on specific rules or conditions (e.g., displaying a "Recommended Products" list based on the user's browsing history).

For example, if you have a *Metrics* component that needs to display the total revenue for the current month, you can use an expression like the following to calculate and display the monthly revenue:

```
{{ui.table1.orders.concat(ui.table1.orderDetails).filter(o => o.orderDate.getMonth() === new Date().getMonth()).reduce((a, b) => a + (b.quantity * b.unitPrice), 0)}}
```

Automation

Select **Automation** as the data source to connect the component to an existing automation or workflow in your application. This is useful when the data or functionality for the component is generated or updated as part of a specific process or workflow.

When to use the **Automation** data source:

- When the data displayed in the component is the result of a specific automation or workflow (e.g., a "Pending Approvals" table that is updated as part of an approval process).
- When you want to trigger actions or updates to the component based on events or conditions within an automation (e.g., updating a Metrics with the latest sales figures for a SKU).
- When you need to integrate the component with other services or systems in your application through an automation (e.g., fetching data from a third-party API and displaying it in a table).

For example, if you have a stepflow component that guides users through a job application process. The stepflow component can be connected to an automation that handles the job application submission, background checks, and offer generation. As the automation progresses through these steps, the stepflow component can dynamically update to reflect the current status of the application.

By carefully selecting the appropriate data source for each component, you can ensure that your application's user interface is powered by the right data and logic, providing a seamless and engaging experience for your users.

Visible if

Use the **Visible if** property to show or hide components or elements based on specific conditions or data values. This is useful when you want to dynamically control the visibility of certain parts of your application's user interface.

The **Visible if** property uses the following syntax:

```
{{expression ? true : false}}
```

or

```
{{expression}}
```

Where *expression* is a boolean expression that evaluates to either `true` or `false`.

If the expression evaluates to `true`, the component will be visible. If the expression evaluates to `false`, the component will be hidden. The expression can reference values from other components, data sources, or variables within your application.

Visible if expression examples

Example: Showing or hiding a password input field based on an email input

Imagine you have a login form with an email input field and a password input field. You want to show the password input field only if the user has entered an email address. You can use the following Visible if expression:

```
{{ui.emailInput.value !== ""}}
```

This expression checks if the value of the `emailInput` component is not an empty string. If the user has entered an email address, the expression evaluates to `true`, and the password input field will be visible. If the email field is empty, the expression evaluates to `false`, and the password input field will be hidden.

Example: Displaying additional form fields based on a dropdown selection

Let's say you have a form where users can select a category from a dropdown list. Depending on the category selected, you want to show or hide additional form fields to gather more specific information.

For example, if the user selects the *Products* category, you can use the following expression to show an additional *Product Details* field:

```
{{ui.categoryDropdown.value === "Products"}}
```

If the user selects the *Services* or *Consulting* categories, you can use this expression to show a different set of additional fields:

```
{{ui.categoryDropdown.value === "Services" || ui.categoryDropdown.value === "Consulting"}}
```

Examples: Other

To make the component visible if the `textInput1` component's value is not an empty string:

```
{{ui.textInput1.value !== "" ? true : false}}
```

To make the component always visible:

```
{{true}}
```

To make the component visible if the `emailInput` component's value is not an empty string:

```
{{ui.emailInput.value !== ""}}
```

Disabled if

The **Disabled if** feature allows you to conditionally enable or disable a component based on specific conditions or data values. This is achieved by using the **Disabled if** property, which accepts a boolean expression that determines whether the component should be enabled or disabled.

The **Disabled if** property uses the following syntax:

```
{{expression ? true : false}}
```

or

```
{{expression}}
```

Disabled if expression examples

Example: Disabling a submit button based on form validation

If you have a form with multiple input fields, and you want to disable the submit button until all required fields are filled out correctly, you can use the following **Disabled If** expression:

```
{{ui.nameInput.value === "" || ui.emailInput.value === "" || ui.passwordInput.value === ""}}
```

This expression checks if any of the required input fields (nameInput, emailInput, passwordInput) are empty. If any of the fields are empty, the expression evaluates to true, and the submit button will be disabled. Once all the required fields are filled out, the expression evaluates to false, and the submit button will be enabled.

By using these types of conditional expressions in the **Visible if** and **Disabled if** properties, you can create dynamic and responsive user interfaces that adapt to user input, providing a more streamlined and relevant experience for your application's users.

Where *expression* is a boolean expression that evaluates to either true or false.

Example:

```
{{ui.textInput1.value === "" ? true : false}}: The component will be Disabled if the textInput1 component's value is an empty string.  
{{!ui.nameInput.isValid || !ui.emailInput.isValid || !ui.passwordInput.isValid}}: The component will be Disabled if any of the named input fields are invalid.
```

Container layouts

The layout properties determine how the content or elements within a component are arranged and positioned. Several layout options are available, each represented by an icon:

- **Column Layout:** This layout arranges the content or elements vertically, in a single column.
- **Two column layout:** This layout divides the component into two equal-width columns, allowing you to position content or elements side by side.
- **Row layout:** This layout arranges the content or elements horizontally, in a single row.

Orientation

- **Horizontal:** This layout arranges the content or elements horizontally, in a single row.
- **Vertical:** This layout arranges the content or elements vertically, in a single column.
- **Inline wrapped:** This layout arranges the content or elements horizontally, but wraps to the next line if the elements exceed the available width.

Alignment

- **Left:** Aligns the content or elements to the left side of the component.
- **Center:** Centers the content or elements horizontally within the component.
- **Right:** Aligns the content or elements to the right side of the component.

Width

The **Width** property specifies the horizontal size of the component. You can enter a percentage value between 0% and 100%, representing the component's width relative to its parent container or the available space.

Height

The **Height** property specifies the vertical size of the component. The "auto" value adjusts the component's height automatically based on its content or the available space.

Space between

The **Space between** property determines the spacing or gap between the content or elements within the component. You can select a value from 0px (no spacing) to 64px, with increments of 4px (e.g., 4px, 8px, 12px, etc.).

Padding

The **Padding** property controls the space between the content or elements and the edges of the component. You can select a value from 0px (no padding) to 64px, with increments of 4px (e.g., 4px, 8px, 12px, etc.).

Background

The **Background** enables or disables a background color or style for the component.

These layout properties provide flexibility in arranging and positioning the content within a component, as well as controlling the size, spacing, and visual appearance of the component itself.

Data components

This section covers the various data components available in the application studio, including the **Table**, **Detail**, **Metric**, **Form**, and **Repeater** components. These components are used to display, gather, and manipulate data within your application.

Table

The **Table** component displays data in a tabular format, with rows and columns. It is used to present structured data, such as lists of items or records from a database, in an organized and easy-to-read manner.

Table properties

The **Table** component shares several common properties with other components, such as Name, Source, and Actions. For more information on these properties, see [Common component properties](#).

In addition to the common properties, the **Table** component has specific properties and configuration options, including Columns, Search and export, and Expressions.

Search and export

The **Table** component provides the following toggles to enable or disable search and export functionality:

- **Show search** When enabled, this toggle adds a search input field to the table, allowing users to search and filter the displayed data.
- **Show export** When enabled, this toggle adds an export option to the table, allowing users to download the table data in various formats, for example: CSV.

Note

By default, the search functionality is limited to the data that has been loaded into the table. To use search exhaustively, you will need to load all pages of data.

Rows per page

You can specify the number of rows to be displayed per page in the table. Users can then navigate between pages to view the full dataset.

Column re-ordering

The **Table** component includes a toggle to enable or disable the ability for users to re-order the columns by dragging and dropping them.

Columns

In this section, you can define the columns to be displayed in the table. Each column can be configured with the following properties:

- **Format:** The data type of the field, for example: text, number, date.
- **Column label:** The header text for the column.
- **Value:** The field from the data source that should be displayed in this column.

This field allows you to specify the value or expression that should be displayed in the column cells. You can use expressions to reference data from the connected source or other components.

Example: `{{currentRow.title}}` - This expression will display the value of the *title* field from the current row in the column cells.

- **Enable sorting:** This toggle allows you to enable or disable sorting functionality for the specific column. When enabled, users can sort the table data based on the values in this column.

Expressions

The **Table** component provides several areas to use expressions and row-level action capabilities that allow you to customize and enhance the table's functionality and interactivity. They allow you to dynamically reference and display data within the table. By leveraging these expression fields, you can create dynamic columns, pass data to row-level actions, and reference table data from other components or expressions within your application.

Examples: Referencing row values

`{{currentRow.columnName}}` or `{{currentRow["Column Name"]}}` These expressions allow you to reference the value of a specific column for the current row being rendered. Replace *columnName* or *Column Name* with the actual name of the column you want to reference.

Examples:

- `{{currentRow.productName}}` Displays the product name for the current row.
- `{{currentRow["Supplier Name"]}}` Displays the supplier name for the current row, where the column header is *Supplier Name*.
- `{{currentRow.orderDate}}` Displays the order date for the current row.

Examples: Referencing selected row

`{{ui.table1.selectedRow["columnName"]}}` This expression allows you to reference the value of a specific column for the currently selected row in the table with the ID *table1*. Replace *table1* with the actual ID of your table component, and *columnName* with the name of the column you want to reference.

Examples:

- `{{ui.ordersTable.selectedRow["totalAmount"]}}` Displays the total amount for the currently selected row in the table with the ID *ordersTable*.
- `{{ui.customersTable.selectedRow["email"]}}` Displays the email address for the currently selected row in the table with the ID *customersTable*.
- `{{ui.employeesTable.selectedRow["department"]}}` Displays the department for the currently selected row in the table with the ID *employeesTable*.

Examples: Creating custom columns

You can add custom columns to a table based on data returned from the underlying data action, automation, or expression. You can use existing column values and JavaScript expressions to create new columns.

Examples:

- `{{currentRow.quantity * currentRow.unitPrice}}` Creates a new column displaying the total price by multiplying the quantity and unit price columns.
- `{{new Date(currentRow.orderDate).toLocaleDateString()}}` Creates a new column displaying the order date in a more readable format.

- `{{currentRow.firstName + ' ' + currentRow.lastName + ' (' + currentRow.email + ')'}}` Creates a new column displaying the full name and email address for each row.

Examples: Customizing column display values:

You can customize the display value of a field within a table column by setting the Value field of the column mapping. This allows you to apply custom formatting or transformations to the displayed data.

Examples:

- `{{ currentRow.rating >= 4 ? '##'.repeat(currentRow.rating) : currentRow.rating }}` Displays star emojis based on the rating value for each row.
- `{{ currentRow.category.toLowerCase().replace(/\b\w/g, c => c.toUpperCase()) }}` Displays the category value with each word capitalized for each row.
- `{{ currentRow.status === 'Active' ? '# Active' : '# Inactive' }}`: Displays a colored circle emoji and text based on the status value for each row.

Row-level button actions

`{{currentRow.columnName}}` or `{{currentRow["Column Name"]}}` You can use these expressions to pass the referenced row's context within a row-level action, such as navigating to another page with the selected row's data or triggering an automation with the row's data.

Examples:

- If you have an edit button in the row action column, you can pass `{{currentRow.orderId}}` as a parameter to navigate to an order editing page with the selected order's ID.
- If you have a delete button in the row action column, you can pass `{{currentRow.customerName}}` to an automation that sends a confirmation email to the customer before deleting their order.
- If you have a view details button in the row action column, you can pass `{{currentRow.employeeId}}` to an automation that logs the employee who viewed the order details.

By leveraging these expression fields and row-level action capabilities, you can create highly customized and interactive tables that display and manipulate data based on your specific requirements. Additionally, you can connect row-level actions with other components or automations within your application, enabling seamless data flow and functionality.

Detail

The **Detail** component is designed to display detailed information about a specific record or item. It provides a dedicated space for presenting comprehensive data related to a single entity or row, making it ideal for showcasing in-depth details or facilitating data entry and editing tasks.

Detail properties

The **Detail** component shares several common properties with other components, such as Name, Source, and Actions. For more information on these properties, see [Common component properties](#).

The **Detail** component also has specific properties and configuration options, including Fields, Layout, and Expressions.

Layout

The **Layout** section allows you to customize the arrangement and presentation of the fields within the **Detail** component. You can configure options such as:

- **Number of columns:** Specify the number of columns to display the fields in.
- **Field ordering:** Drag and drop fields to reorder their appearance.
- **Spacing and alignment:** Adjust the spacing and alignment of fields within the component.

Expressions and examples

The **Detail** component provides various expression fields that allow you to reference and display data within the component dynamically. These expressions enable you to create customized and interactive **Detail** components that seamlessly connect with your application's data and logic.

Example: Referencing data

`{{ui.details.data[0]?["colName"]}}`: This expression allows you to reference the value of the column named "colName" for the first item (index 0) in the data array connected to the **Detail** component with the ID "details". Replace "colName" with the actual name of the column you want

to reference. For example, the following expression will display the value of the "customerName" column for the first item in the data array connected to the "details" component:

```
{{ui.details.data[0]?."customerName"}}
```

Note

This expression is useful when the **Detail** component is on the same page as the table being referenced, and you want to display data from the first row of the table in the **Detail** component.

Example: Conditional rendering

`{{ui.table1.selectedRow["colName"]}}`: This expression returns true if the selected row in the table with the ID *table1* has data for the column named *colName*. It can be used to conditionally show or hide the **Detail** component based on whether the table's selected row is empty or not.

Example:

You can use this expression in the `Visible if` property of the **Detail** component to conditionally show or hide it based on the selected row in the table.

```
{{ui.table1.selectedRow["customerName"]}}
```

If this expression evaluates to true (the selected row in the *table1* component has a value for the *customerName* column), the **Detail** component will be visible. If the expression evaluates to false (i.e., the selected row is empty or does not have a value for "customerName"), the **Detail** component will be hidden.

Example: Conditional display

`{{(ui.Component.value === "green" ? "#" : ui.Component.value === "yellow" ? "#" : ui.detail1.data?.[0]?.CustomerStatus)}}`: This expression conditionally displays an emoji based on the value of a component or data field.

Breakdown:

- `ui.Component.value`: References the value of a component with the ID *Component*.

- `=== "green"`: Checks if the component's value is equal to the string "green".
- `? "#"`: If the condition is true, displays the green circle emoji.
- `: ui.Component.value === "yellow" ? "#"`: If the first condition is false, checks if the component's value is equal to the string "yellow".
- `? "#"`: If the second condition is true, displays the yellow square emoji.
- `: ui.detail1.data?.[0]?.CustomerStatus`: If both conditions are false, it references the "CustomerStatus" value of the first item in the data array connected to the Detail component with the ID "detail1".

This expression can be used to display an emoji or a specific value based on the value of a component or data field within the **Detail** component.

Metrics

The **Metrics** component is a visual element that displays key metrics or data points in a card-like format. It is designed to provide a concise and visually appealing way to present important information or performance indicators.

Metrics properties

The **Metrics** component shares several common properties with other components, such as Name, Source, and Actions. For more information on these properties, see [Common component properties](#).

Trend

The Metrics's trend feature allows you to display a visual indicator of the performance or change over time for the metric being displayed.

Trend value

This field allows you to specify the value or expression that should be used to determine the trend direction and magnitude. Typically, this would be a value that represents the change or performance over a specific time period.

Example:

```
{{ui.salesMetrics.data?.[0]?.monthOverMonthRevenue}}
```

This expression retrieves the month-over-month revenue value from the first item in the data connected to the "salesMetrics" Metrics.

Positive trend

This field allows you to enter an expression that defines the condition for a positive trend. The expression should evaluate to true or false.

Example:

```
{{ui.salesMetrics.data?.[0]?.monthOverMonthRevenue > 0}}
```

This expression checks if the month-over-month revenue value is greater than 0, indicating a positive trend.

Negative trend

This field allows you to enter an expression that defines the condition for a negative trend. The expression should evaluate to true or false.

Example:

```
{{ui.salesMetrics.data?.[0]?.monthOverMonthRevenue < 0}}
```

This expression checks if the month-over-month revenue value is less than 0, indicating a negative trend.

Color bar

This toggle allows you to enable or disable the display of a colored bar to visually indicate the trend status.

Color Bar examples:

Example: Sales metrics trend

- **Trend value:** `{{ui.salesMetrics.data?.[0]?.monthOverMonthRevenue}}`
- **Positive trend:** `{{ui.salesMetrics.data?.[0]?.monthOverMonthRevenue > 0}}`
- **Negative trend:** `{{ui.salesMetrics.data?.[0]?.monthOverMonthRevenue < 0}}`
- **Color bar:** Enabled

Example: inventory metrics trend

- **Trend value:** `{{ui.inventoryMetrics.data?.[0]?.currentInventory - ui.inventoryMetrics.data?.[1]?.currentInventory}}`
- **Positive trend:** `{{ui.inventoryMetrics.data?.[0]?.currentInventory > ui.inventoryMetrics.data?.[1]?.currentInventory}}`
- **Negative trend:** `{{ui.inventoryMetrics.data?.[0]?.currentInventory < ui.inventoryMetrics.data?.[1]?.currentInventory}}`
- **Color Bbar:** Enabled

Example: Customer satisfaction trend

- **Trend value:** `{{ui.customerSatisfactionMetrics.data?.[0]?.npsScore}}`
- **Positive trend:** `{{ui.customerSatisfactionMetrics.data?.[0]?.npsScore >= 8}}`
- **Negative trend:** `{{ui.customerSatisfactionMetrics.data?.[0]?.npsScore < 7}}`
- **Color bar:** Enabled

By configuring these trend-related properties, you can create **Metrics** components that provide a visual representation of the performance or change over time for the metric being displayed.

By leveraging these expressions, you can create highly customized and interactive metrics components that reference and display data dynamically, allowing you to showcase key metrics, performance indicators, and data-driven visualizations within your application.

Metrics expression examples

In the properties panel, you can enter expressions to display the title, primary value, secondary value, and value caption to dynamically display a value.

Example: Referencing primary value

`{{ui.metric1.primaryValue}}`: This expression allows you to reference the primary value of the **Metrics** component with the ID *metric1* from other components or expressions within the same page.

Example: `{{ui.salesMetrics.primaryValue}}` will display the primary value of the *salesMetrics* **Metrics** component.

Example: Referencing secondary value

`{{ui.metric1.secondaryValue}}`: This expression allows you to reference the secondary value of the **Metrics** component with the ID *metric1* from other components or expressions within the same page.

Example: `{{ui.revenueMetrics.secondaryValue}}` will display the secondary value of the *revenueMetrics* **Metrics** component.

Example: Referencing data

`{{ui.metric1.data}}`: This expression allows you to reference the data of the **Metrics** component with the ID *metric1* from other components or expressions within the same page.

Example: `{{ui.kpiMetrics.data}}` will reference the data connected to the *kpiMetrics* **Metrics** component.

Example: Displaying specific data values:

`{{ui.metric1.data?.[0]?.id}}`: This expression is an example of how to display a specific piece of information from the data connected to the **Metrics** component with the ID *metric1*. It is useful when you want to display a specific property of the first item in the data.

Breakdown:

- `ui.metric1`: References the **Metrics** component with the ID *metric1*.
- `data`: Refers to the information or data set connected to that component.
- `?.[0]`: Means the first item or entry in that data set.
- `?.id`: Displays the *id* value or identifier of that first item or entry.

Example: `{{ui.orderMetrics.data?.[0]?.orderId}}` will display the *orderId* value of the first item in the data connected to the *orderMetrics* **Metrics** component.

Example: Displaying data length

`{{ui.metric1.data?.length}}`: This expression demonstrates how to display the length (number of items) in the data connected to the **Metrics** component with the ID *metric1*. It is useful when you want to display the number of items in the data.

Breakdown:

- `ui.metric1.data`: References the data set connected to the component.
- `?.length`: Accesses the total count or number of items or entries in that data set.

Example: `{{ui.productMetrics.data?.length}}` will display the number of items in the data connected to the *productMetrics* **Metrics** component.

Repeater

The **Repeater** component is a dynamic component that allows you to generate and display a collection of elements based on a provided data source. It is designed to facilitate the creation of lists, grids, or repeating patterns within your application's user interface. A few example use cases include:

- Displaying a card for each user in an account
- Showing a list of products that include images and a button to add it to the cart
- Showing a list of files the user can access

The **Repeater** component differentiates itself from the **Table** component with rich content. A **Table** component has a strict row and column format. The **Repeater** can display your data more flexibly.

Repeater properties

The **Repeater** component shares several common properties with other components, such as Name, Source, and Actions. For more information on these properties, see [Common component properties](#).

In addition to the common properties, the **Repeater** component has the following additional properties and configuration options.

Item template

The **Item template** is a container where you can define the structure and components that will be repeated for each item in the data source. You can drag and drop other components into this container, such as **Text**, **Image**, **Button**, or any other component you need to represent each item.

Within the **Item template**, you can reference properties or values from the current item using expressions in the format `{{currentItem.propertyName}}`.

For example, if your data source contains an `itemName` property, you can use `{{currentItem.itemName}}` to display the item name(s) of the current item.

Layout

The **Layout** section allows you to configure the arrangement of the repeated elements within the Repeater Component.

Orientation

- **List:** Arranges the repeated elements vertically in a single column.
- **Grid:** Arranges the repeated elements in a grid layout with multiple columns.

Rows per page

Specify the number of rows to display per page in the list layout. Pagination is provided for items that overflow the specified number of rows.

Columns and Rows per Page (Grid)

- **Columns:** Specify the number of columns in the grid layout.
- **Rows per Page:** Specify the number of rows to display per page in the grid layout. Pagination is provided for items that overflow the specified grid dimensions.

Expressions and examples

The **Repeater** component provides various expression fields that allow you to reference and display data within the component dynamically. These expressions enable you to create customized and interactive **Repeater** components that seamlessly connect with your application's data and logic.

Example: Referencing items

- `{{currentItem.propertyName}}`: Reference properties or values from the current item within the **Item Template**.
- `{{ui.repeaterID[index]}}`: Reference a specific item in the Repeater Component by its index.

Example: Rendering a list of products

- **Source:** Select the *Products* entity as the data source.

- **Item Template:** Add a **Container** component with a **Text** component inside to display the product name (`{{currentItem.productName}}`) and an **Image** component to display the product image (`{{currentItem.productImageUrl}}`).
- **Layout:** Set the `Orientation` to `List` and adjust the `Rows per Page` as desired.

Example: Generating a grid of user avatars

- **Source:** Use an expression to generate an array of user data (e.g., `[{name: 'John', avatarUrl: '...'}, {...}, {...}]`).
- **Item Template:** Add an **Image** component and set its `Source` property to `{{currentItem.avatarUrl}}`.
- **Layout:** Set the `Orientation` to `Grid`, specify the number of `Columns` and `Rows per Page`, and adjust the `Space Between` and `Padding` as needed.

By using the `Repeater` component, you can create dynamic and data-driven user interfaces, streamlining the process of rendering collections of elements and reducing the need for manual repetition or hard-coding.

Form

The `Form` component is designed to capture user input and facilitate data entry tasks within your application. It provides a structured layout for displaying input fields, dropdowns, checkboxes, and other form controls, allowing users to input or modify data seamlessly. You can nest other components inside of a form component, such as a table.

Form properties

The `Form` component shares several common properties with other components, such as `Name`, `Source`, and `Actions`. For more information on these properties, see [Common component properties](#).

Generate Form

The **Generate Form** feature makes it easy to quickly create form fields by automatically populating them based on a selected data source. This can save time and effort when building forms that need to display a large number of fields.

To use the Generate Form feature:

1. In the **Form** component's properties, locate the **Generate Form** section.
2. Select the data source you want to use to generate the form fields. This can be an entity, workflow, or any other data source available in your application.
3. The form fields will be automatically generated based on the selected data source, including the field labels, types, and data mappings.
4. Review the generated fields and make any necessary customizations, such as adding validation rules or changing the field order.
5. Once you're satisfied with the form configuration, choose **Submit** to apply the generated fields to your **Form** component.

The **Generate Form** feature is particularly useful when you have a well-defined data model or set of entities in your application that you need to capture user input for. By automatically generating the form fields, you can save time and ensure consistency across your application's forms.

After using the **Generate Form** feature, you can further customize the layout, actions, and expressions for the **Form** component to fit your specific requirements.

Expressions and examples

Like other components, you can use expressions to reference and display data within the **Form** component. For example:

- `{{ui.userForm.data.email}}`: References the value of the `email` field from the data source connected to the **Form** component with the ID `userForm`.

Note

See [Common component properties](#) for more expression examples of the common properties.

By configuring these properties and leveraging expressions, you can create customized and interactive Form components that seamlessly integrate with your application's data sources and logic. These components can be used to capture user input, display pre-populated data, and trigger actions based on the form submissions or user interactions.

Stepflow

The **Stepflow** component is designed to guide users through multi-step processes or workflows within your application. It provides a structured and intuitive interface for presenting a sequence of steps, each with its own set of inputs, validations, and actions.

The Stepflow component shares several common properties with other components, such as Name, Source, and Actions. For more information on these properties, see [Common component properties](#).

The **Stepflow** component has additional properties and configuration options, such as Step Navigation, Validation, and Expressions.

Text & number components

Text input

The **Text input** component allows users to enter and submit text data within your application. It provides a simple and intuitive way to capture user input, such as names, addresses, or any other textual information.

- `{{ui.inputTextID.value}}`: Returns the provided value in the input field.
- `{{ui.inputTextID.isValid}}`: Returns the validity of the provided value in the input field.

Text

The **Text** component is used to display textual information within your application. It can be used to show static text, dynamic values, or content generated from expressions.

Text area

The **Text area** component is designed to capture multi-line text input from users. It provides a larger input field area for users to enter longer text entries, such as descriptions, notes, or comments.

- `{{ui.textAreaID.value}}`: Returns the provided value in the text area.
- `{{ui.textAreaID.isValid}}`: Returns the validity of the provided value in the text area.

Email

The **Email** component is a specialized input field designed to capture email addresses from users. It can enforce specific validation rules to ensure the entered value adheres to the correct email format.

- `{{ui.emailID.value}}`: Returns the provided value in the email input field.
- `{{ui.emailID.isValid}}`: Returns the validity of the provided value in the email input field.

Password

The **Password** component is an input field specifically designed for users to enter sensitive information, such as passwords or PIN codes. It masks the entered characters to maintain privacy and security.

- `{{ui.passwordID.value}}`: Returns the provided value in the password input field.
- `{{ui.passwordID.isValid}}`: Returns the validity of the provided value in the password input field.

Search

The **Search** component provides users with a dedicated input field for performing search queries or entering search terms within the populated data within the application.

- `{{ui.searchID.value}}`: Returns the provided value in the search field.

Phone

The **Phone** component is an input field tailored for capturing phone numbers or other contact information from users. It can include specific validation rules and formatting options to ensure the entered value adheres to the correct phone number format.

- `{{ui.phoneID.value}}`: Returns the provided value in the phone input field.
- `{{ui.phoneID.isValid}}`: Returns the validity of the provided value in the phone input field.

Number

The **Number** component is an input field designed specifically for users to enter numerical values. It can enforce validation rules to ensure the entered value is a valid number within a specified range or format.

- `{{ui.numberID.value}}`: Returns the provided value in the number input field.
- `{{ui.numberID.isValid}}`: Returns the validity of the provided value in the number input field.

Currency

The **Currency** component is a specialized input field for capturing monetary values or amounts. It can include formatting options to display currency symbols, decimal separators, and enforce validation rules specific to currency inputs.

- `{{ui.currencyID.value}}`: Returns the provided value in the currency input field.
- `{{ui.currencyID.isValid}}`: Returns the validity of the provided value in the currency input field.

Switch

The **Switch** component is a user interface control that allows users to toggle between two states or options, such as on/off, true/false, or enabled/disabled. It provides a visual representation of the current state and allows users to change it with a single click or tap.

Detail pair

The **Detail pair** component is used to display key-value pairs or pairs of related information in a structured and readable format. It is commonly used to present details or metadata associated with a specific item or entity.

Selection components

Switch group

The **Switch group** component is a collection of individual switch controls that allow users to select one or more options from a predefined set. It provides a visual representation of the selected and unselected options, making it easier for users to understand and interact with the available choices.

Switch group expression fields

- `{{ui.switchGroupID.value}}`: Returns an array of strings containing the value of each switch that is enabled by the app user.

Checkbox group

The **Checkbox group** component presents users with a group of checkboxes, allowing them to select multiple options simultaneously. It is useful when you want to provide users with the ability to choose one or more items from a list of options.

Checkbox group expression fields

- `{{ui.checkboxGroupID.value}}`: Returns an array of strings containing the value of each checkbox that is selected by the app user.

Radio group

The **Radio group** component is a set of radio buttons that allow users to select a single option from multiple mutually exclusive choices. It ensures that only one option can be selected at a time, providing a clear and unambiguous way for users to make a selection.

Radio group expression fields

The following fields can be used in expressions.

- `{{ui.radioGroupID.value}}`: Returns the value of the radio button that is selected by the app user.

Single select

The **Single select** component presents users with a list of options, from which they can select a single item. It is commonly used in scenarios where users need to make a choice from a predefined set of options, such as selecting a category, a location, or a preference.

Single select expression fields

- `{{ui.singleSelectID.value}}`: Returns the value of the list item that is selected by the app user.

Multi select

The **Multi select** component is similar to the **Single select** component but allows users to select multiple options simultaneously from a list of choices. It is useful when users need to make multiple selections from a predefined set of options, such as selecting multiple tags, interests, or preferences.

Multi select expression fields

- `{{ui.multiSelectID.value}}`: Returns an array of strings containing the value of each list item that is selected by the app user.

Buttons & navigation components

The application studio provides a variety of button and navigation components to allow users to trigger actions and navigate within your application.

Button components

The available button components are:

- Button
- Outlined button
- Icon button
- Text button

These button components share the following common properties:

Content

- **Button label:** The text to be displayed on the button.

Type

- **Button:** A standard button.
- **Outlined:** A button with an outlined style.
- **Icon:** A button with an icon.

- **Text:** A text-only button.

Size

The size of the button. Possible values are Small, Medium, and Large.

Icon

You can select from a variety of icons to be displayed on the button, including:

- Envelope Closed
- Bell
- Person
- Hamburger Menu
- Search
- Info Circled
- Gear
- Chevron Left
- Chevron Right
- Dots Horizontal
- Trash
- Edit
- Check
- Close
- Home
- Plus

Triggers

When the button is clicked, you can configure one or more actions to be triggered. The available action types are:

- **Basic**
 - Run component action: Executes a specific action within a component.

- **Navigate:** Navigates to another page or view.
- **Invoke Data Action:** Triggers a data-related action, such as creating, updating, or deleting a record.
- **Advanced**
 - **JavaScript:** Runs custom JavaScript code.
 - **Invoke Automation:** Starts an existing automation or workflow.

JavaScript action button properties

Select the JavaScript action type to run custom JavaScript code when the button is clicked.

Source code

In the `Source code` field, you can enter your JavaScript expression or function. For example:

```
return "Hello World";
```

This will simply return the string `Hello World` when the button is clicked.

Condition: Run if

You can also provide a boolean expression that determines whether the JavaScript action should be executed or not. This uses the following syntax:

```
{{ui.textinput1.value !== ""}}
```

In this example, the JavaScript action will only run if the value of the `textInput1` component is not an empty string.

By using these advanced trigger options, you can create highly customized button behaviors that integrate directly with your application's logic and data. This allows you to extend the built-in functionality of the buttons and tailor the user experience to your specific requirements.

Note

Always thoroughly test your JavaScript actions to ensure they are functioning as expected.

Hyperlink

The **Hyperlink** component provides a clickable link for navigating to external URLs or internal application routes.

Hyperlink properties

Content

- **Hyperlink label:** The text to be displayed as the hyperlink label.

URL

The destination URL for the hyperlink, which can be an external website or an internal application route.

Triggers

When the hyperlink is clicked, you can configure one or more actions to be triggered. The available action types are the same as those for the button components.

Date & time components

Date

The **Date** component allows users to select and input dates.

The **Date** component shares several common properties with other components, such as Name, Source, and Validation. For more information on these properties, see [Common component properties](#).

In addition to the common properties, the **Date** component has the following specific properties:

Date properties

Format

- **YYYY/MM/DD, DD/MM/YYYY, YYYY/MM/DD, YYYY/DD/MM, MM/DD, DD/MM:** The format in which the date should be displayed.

Value

- **YYYY-MM-DD:** The format in which the date value is stored internally.

Min date

- **YYYY-MM-DD**: The minimum date that can be selected.

Note

This value must match the format of YYYY-MM-DD.

Max date

- **YYYY-MM-DD**: The maximum date that can be selected.

Note

This value must match the format of YYYY-MM-DD.

Calendar type

- **1 Month, 2 Months**: The type of calendar UI to display.

Disabled dates

- **Source**: The data source for the dates that should be disabled. For example: None, Expression.
- **Disabled dates**: An expression that determines which dates should be disabled, such as:
 - `{{currentRow.column}}`: Disables dates that match what this expression evaluates to.
 - `{{new Date(currentRow.dateColumn) < new Date("2023-01-01")}}`: Disables dates before January 1, 2023
 - `{{new Date(currentRow.dateColumn).getDay() === 0 || new Date(currentRow.dateColumn).getDay() === 6}}`: Disables weekends.

Behavior

- **Visible if**: An expression that determines the visibility of the **Date** component.
- **Disable if**: An expression that determines whether the **Date** component should be disabled.

Validation

The **Validation** section allows you to define additional rules and constraints for the date input. By configuring these validation rules, you can ensure that the date values entered by users meet the specific requirements of your application. You can add the following types of validations:

- **Required:** This toggle ensures that the user must enter a date value before submitting the form.
- **Custom:** You can create custom validation rules using JavaScript expressions. For example:

```
{{new Date(ui.dateInput.value) < new Date("2023-01-01")}}
```

This expression checks if the entered date is before January 1, 2023. If the condition is true, the validation will fail.

You can also provide a custom validation message to be displayed when the validation is not met:

```
"Validation not met. The date must be on or after January 1, 2023."
```

By configuring these validation rules, you can ensure that the date values entered by users meet the specific requirements of your application.

Expressions and examples

The **Date** component provides the following expression field:

- `{{ui.dateID.value}}`: Returns the date value entered by the user in the format YYYY-MM-DD.

Time

The **Time** component allows users to select and input time values. By configuring the various properties of the **Time** component, you can create time input fields that meet the specific requirements of your application, such as restricting the selectable time range, disabling certain times, and controlling the component's visibility and interactivity.

Time properties

The **Time** component shares several common properties with other components, such as `Name`, `Source`, and `Validation`. For more information on these properties, see [Common component properties](#).

In addition to the common properties, the **Time** component has the following specific properties:

Time intervals

- **5 minutes, 10 minutes, 15 minutes, 20 minutes, 25 minutes, 30 minutes, 60 minutes:** The intervals available for selecting the time.

Value

- **HH:MM AA:** The format in which the time value is stored internally.

Note

This value must match the format of HH:MM AA.

Placeholder

- **Calendar settings:** The placeholder text displayed when the time field is empty.

Min time

- **HH:MM AA:** The minimum time that can be selected.

Note

This value must match the format of HH:MM AA.

Max time

- **HH:MM AA:** The maximum time that can be selected.

Note

This value must match the format of HH:MM AA.

Disabled times

- **Source:** The data source for the times that should be disabled (e.g., None, Expression).
- **Disabled times:** An expression that determines which times should be disabled, such as `{{currentRow.column}}`.

Disabled times configuration

You can use the **Disabled Times** section to specify which time values should be unavailable for selection.

Source

- **None:** No times are disabled.
- **Expression:** You can use a JavaScript expression to determine which times should be disabled, such as `{{currentRow.column}}`.

Example expression:

```
{{currentRow.column === "Lunch Break"}}
```

This expression would disable any times where the "Lunch Break" column is true for the current row.

By configuring these validation rules and disabled time expressions, you can ensure that the time values entered by users meet the specific requirements of your application.

Behavior

- **Visible if:** An expression that determines the visibility of the Time component.
- **Disable if:** An expression that determines whether the Time component should be disabled.

Validation

- **Required:** A toggle that ensures the user must enter a time value before submitting the form.
- **Custom:** Allows you to create custom validation rules using JavaScript expressions.

Custom Validation Message: The message to be displayed when the custom validation is not met.

For example:

```
{{ui.timeInput.value === "09:00 AM" || ui.timeInput.value === "09:30 AM"}}
```

This expression checks if the entered time is 9:00 AM or 9:30 AM. If the condition is true, the validation will fail.

You can also provide a custom validation message to be displayed when the validation is not met:

```
Validation not met. The time must be 9:00 AM or 9:30 AM.
```

Expressions and examples

The Time component provides the following expression field:

- `{{ui.timeID.value}}`: Returns the time value entered by the user in the format HH:MM AA.

Example: Time value

- `{{ui.timeID.value}}`: Returns the time value entered by the user in the format HH:MM AA.

Example: Time comparison

- `{{ui.timeInput.value > "10:00 AM"}}`: Checks if the time value is greater than 10:00 AM.
- `{{ui.timeInput.value < "05:00 PM"}}`: Checks if the time value is less than 05:00 PM.

Date range

The **Date range** component allows users to select and input a range of dates. By configuring the various properties of the Date Range component, you can create date range input fields that meet the specific requirements of your application, such as restricting the selectable date range, disabling certain dates, and controlling the component's visibility and interactivity.

Date range properties

The **Date Range** component shares several common properties with other components, such as Name, Source, and Validation. For more information on these properties, see [Common component properties](#).

In addition to the common properties, the **Date Range** component has the following specific properties:

Format

- **MM/DD/YYYY**: The format in which the date range should be displayed.

Start date

- **YYYY-MM-DD**: The minimum date that can be selected as the start of the range.

Note

This value must match the format of YYYY-MM-DD.

End date

- **YYYY-MM-DD**: The maximum date that can be selected as the end of the range.

Note

This value must match the format of YYYY-MM-DD.

Placeholder

- **Calendar settings:** The placeholder text displayed when the date range field is empty.

Min date

- **YYYY-MM-DD:** The minimum date that can be selected.

Note

This value must match the format of YYYY-MM-DD.

Max date

- **YYYY-MM-DD:** The maximum date that can be selected.

Note

This value must match the format of YYYY-MM-DD.

Calendar type

- **1 Month:** The type of calendar UI to display. For example, single month.
- **2 Month:** The type of calendar UI to display. For example, two months.

Mandatory days selected

- **0:** The number of mandatory days that must be selected within the date range.

Disabled dates

- **Source:** The data source for the dates that should be disabled (e.g., None, Expression, Entity, or Automation).
- **Disabled dates:** An expression that determines which dates should be disabled, such as `{{currentRow.column}}`.

Validation

The **Validation** section allows you to define additional rules and constraints for the date range input.

Expressions and examples

The **Date Range** component provides the following expression fields:

- `{{ui.dateRangeID.startDate}}`: Returns the start date of the selected range in the format YYYY-MM-DD.
- `{{ui.dateRangeID.endDate}}`: Returns the end date of the selected range in the format YYYY-MM-DD.

Example: Calculating date difference

- `{{(new Date(ui.dateRangeID.endDate) - new Date(ui.dateRangeID.startDate)) / (1000 * 60 * 60 * 24)}}` Calculates the number of days between the start and end dates.

Example: Conditional visibility based on date range

- `{{new Date(ui.dateRangeID.startDate) < new Date("2023-01-01") || new Date(ui.dateRangeID.endDate) > new Date("2023-12-31")}}` Checks if the selected date range is outside of the year 2023.

Example: Disabled dates based on current row data

- `{{currentRow.isHoliday}}` Disables dates where the "isHoliday" column in the current row is true.
- `{{new Date(currentRow.dateColumn) < new Date("2023-01-01")}}` Disables dates before January 1, 2023 based on the "dateColumn" in the current row.
- `{{new Date(currentRow.dateColumn).getDay() === 0 || new Date(currentRow.dateColumn).getDay() === 6}}` Disables weekends based on the "dateColumn" in the current row.

Custom validation

- `{{new Date(ui.dateRangeID.startDate) > new Date(ui.dateRangeID.endDate)}}`
Checks if the start date is later than the end date, which would fail the custom validation.

Media components

The application studio provides several components for embedding and displaying various media types within your application.

iFrame embed

The **iFrame embed** component allows you to embed external web content or applications within your application using an iFrame.

iFrame embed properties

URL

The URL of the external content or application you want to embed.

Layout

- **Width:** The width of the iFrame, specified as a percentage (%) or a fixed pixel value (e.g., 300px).
- **Height:** The height of the iFrame, specified as a percentage (%) or a fixed pixel value.

S3 upload

The **S3 upload** component allows users to upload files to an Amazon S3 bucket. By configuring the **S3 Upload** component, you can enable users to easily upload files to your application's Amazon S3 storage, and then leverage the uploaded file information within your application's logic and user interface.

Note

Remember to ensure that the necessary permissions and Amazon S3 bucket configurations are in place to support the file uploads and storage requirements of your application.

S3 upload properties

S3 Configuration

- **Connector:** Select the pre-configured Amazon S3 connector to use for the file uploads.
- **Bucket:** The Amazon S3 bucket where the files will be uploaded.
- **Folder:** The folder within the Amazon S3 bucket where the files will be stored.
- **File name:** The naming convention for the uploaded files.

File upload configuration

- **Label:** The label or instructions displayed above the file upload area.
- **Description:** Additional instructions or information about the file upload.
- **File type:** The type of files that are allowed to be uploaded. For example: image, document, or video.
- **Size:** The maximum size of the individual files that can be uploaded.
- **Button label:** The text displayed on the file selection button.
- **Button style:** The style of the file selection button. For example, outlined or filled.
- **Button size:** The size of the file selection button.

Validation

- **Max number of files:** The maximum number of files that can be uploaded at once.
- **Max file size:** The maximum size allowed for each individual file.

Triggers

- **On success:** Actions to be triggered when a file upload is successful.
- **On failure:** Actions to be triggered when a file upload fails.

S3 upload expression fields

The **S3 upload** component provides the following expression fields:

- `{{ui.s3uploadID.files}}`: Returns an array of the files that have been uploaded.

- `{{ui.s3uploadID.files[0]?.size}}`: Returns the size of the file at the designated index.
- `{{ui.s3uploadID.files[0]?.type}}`: Returns the type of the file at the designated index.
- `{{ui.s3uploadID.files[0]?.nameOnly}}`: Returns the name of the file, with no extension suffix, at the designated index.
- `{{ui.s3uploadID.files[0]?.nameWithExtension}}`: Returns the name of the file with its extension suffix at the designated index.

Expressions and examples

Example: Accessing uploaded files

- `{{ui.s3uploadID.files.length}}`: Returns the number of files that have been uploaded.
- `{{ui.s3uploadID.files.map(f => f.name).join(', ')}}`: Returns a comma-separated list of the file names that have been uploaded.
- `{{ui.s3uploadID.files.filter(f => f.type.startsWith('image/'))}}`: Returns an array of only the image files that have been uploaded.

Example: Validating file uploads

- `{{ui.s3uploadID.files.some(f => f.size > 5 * 1024 * 1024)}}`: Checks if any of the uploaded files exceed 5 MB in size.
- `{{ui.s3uploadID.files.every(f => f.type === 'image/png')}}`: Checks if all the uploaded files are PNG images.
- `{{ui.s3uploadID.files.length > 3}}`: Checks if more than 3 files have been uploaded.

Example: Triggering actions

- `{{ui.s3uploadID.files.length > 0 ? 'Upload Successful' : 'No files uploaded'}}`: Displays a success message if at least one file has been uploaded.
- `{{ui.s3uploadID.files.some(f => f.type.startsWith('video/')) ? triggerVideoProcessing() : null}}`: Triggers a video processing automation if any video files have been uploaded.
- `{{ui.s3uploadID.files.map(f => f.url)}}`: Retrieves the URLs of the uploaded files, which can be used to display or further process the files.

These expressions allow you to access the uploaded files, validate the file uploads, and trigger actions based on the file upload results. By utilizing these expressions, you can create more dynamic and intelligent behavior within your application's file upload functionality.

Note

Replace *s3uploadID* with the ID of your **S3 upload** component.

PDF viewer component

The **PDF viewer** component allows users to view and interact with PDF documents within your application. App Studio supports these different input types for the PDF Source, the **PDF viewer** component provides flexibility in how you can integrate PDF documents into your application, whether it's from a static URL, an inline data URI, or dynamically generated content.

PDF viewer properties

Source

The source of the PDF document, which can be an expression, entity, URL, or automation.

Expression

Use an expression to dynamically generate the PDF source.

Entity

Connect the **PDF viewer** component to a data entity that contains the PDF document.

URL

Specify the URL of the PDF document.

URL

You can enter a URL that points to the PDF document you want to display. This could be a public web URL or a URL within your own application.

Example: `https://example.com/document.pdf`

Data URI

A **Data URI** is a compact way to include small data files (like images or PDFs) inline within your application. The PDF document is encoded as a base64 string and included directly in the component's configuration.

Blob or ArrayBuffer

You can also provide the PDF document as a Blob or ArrayBuffer object, which allows you to dynamically generate or retrieve the PDF data from various sources within your application.

Automation

Connect the **PDF viewer** component to an automation that provides the PDF document.

Actions

- **Download:** Adds a button or link that allows users to download the PDF document.

Layout

- **Width:** The width of the PDF Viewer, specified as a percentage (%) or a fixed pixel value (e.g., 600px).
- **Height:** The height of the PDF Viewer, specified as a fixed pixel value.

Image viewer

The **Image viewer** component allows users to view and interact with image files within your application.

Image viewer properties

Source

- **Entity:** Connect the **Image viewer** component to a data entity that contains the image file.
- **URL:** Specify the URL of the image file.
- **Expression:** Use an expression to dynamically generate the image source.
- **Automation:** Connect the **Image viewer** component to an automation that provides the image file.

Alt text

The alternative text description of the image, which is used for accessibility purposes.

Layout

- **Image fit:** Determines how the image should be resized and displayed within the component. For example: `Contain`, `Cover`, or `Fill`.
- **Width:** The width of the **Image viewer** component, specified as a percentage (%) or a fixed pixel value (e.g., `300px`).
- **Height:** The height of the **Image viewer** component, specified as a fixed pixel value.
- **Background:** Allows you to set a background color or image for the **Image viewer** component.

Defining and implementing your app's business logic with automations

Built in the application studio, **automations** are how you define the business logic of your application. The main components of an automation are: triggers that start the automation, a sequence of one or more actions, input parameters used to pass data to the automation, and an output.

Topics

- [Automations concepts](#)
- [Tutorial: Interacting with Amazon Simple Storage Service using automations](#)
- [Creating, editing, and deleting automations](#)
- [Adding, editing, and deleting automation actions](#)
- [Automation actions reference](#)

Automations concepts

Here are some concepts and terms to know when defining and configuring your app's business logic using automations in App Studio.

Automations

Built in the application studio, **automations** are how you define the business logic of your application. The main components of an automation are: triggers that start the automation, a sequence of one or more actions, input parameters used to pass data to the automation, and an output.

Actions

An automation action, commonly referred to as an **action**, is an individual step of logic that make up an automation. Each action performs a specific task, whether it's sending an email, creating a data record, invoking a Lambda function, or calling APIs. Actions are added to automations from the action library, and can be grouped into conditional statements or loops.

Automation input parameters

Automation input parameters are dynamic input values that you can pass in from components to automations to make them flexible and reusable. Think of parameters as variables for your automation, instead of hard-coding values into an automation, you can define parameters and provide different values when needed. Parameters allow you to use the same automation with different inputs each time it is run.

Mocked output

Some actions interact with external resources or services using connectors. When using the preview environment, applications do not interact with external services. To test actions that use connectors in the preview environment, you can use **mocked output** to simulate the connector's behavior and output. The mocked output is configured using JavaScript, and the result is stored in an action's results, just as the connector's response is stored in a published app.

By using mocking, you can use the preview environment to test various scenarios and their impact on other actions with the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without calling the external service through connectors.

Automation output

An **automation output** is used to pass values from one automation to other resources of an app, such as components or other automations. Automation outputs are configured as expressions, and the expression can return a static value or a dynamic value computed from automation parameters

and actions. By default, automations do not return any data, including the results of actions within the automation.

A couple of examples of how automation outputs can be used:

- You can configure an automation output to return an array, and pass that array to populate a data component.
- You can use an automation to calculate a value, and pass that value into multiple other automations as a way to centralize and reuse business logic.

Triggers

Trigger determine when, and on what conditions, an automation will run. Some examples of triggers are `On click` for buttons and `On select` for text inputs. The type of component determines the list of available triggers for that component. Triggers are added to [components](#) and configured in the application studio.

Tutorial: Interacting with Amazon Simple Storage Service using automations

You can invoke various Amazon S3 operations from an App Studio app, for example, you could create a simple admin panel to manage your users and orders and display your media from Amazon S3. While you can invoke any Amazon S3 operation using the **Invoke AWS** action, there are four dedicated Amazon S3 actions that you can add to automations in your app to perform common operations on Amazon S3 buckets and objects. The four actions and their operations are as follows:

- **Put Object:** Uses the Amazon S3 `PutObject` operation to add an object an Amazon S3 bucket.
- **Get Object:** Uses the Amazon S3 `GetObject` operation to retrieve an object from an Amazon S3 bucket.
- **List Objects:** Uses the Amazon S3 `ListObjects` operation to list objects in an Amazon S3 bucket.
- **Delete Object:** Uses the Amazon S3 `DeleteObject` operation to delete an object from an Amazon S3 bucket.

In addition to the actions, there is an **S3 upload** component that can be added to pages in applications. Users can use this component to choose a file to upload, and the component calls

Amazon S3 PutObject to upload the file to the configured bucket and folder. This tutorial will use this component in place of the standalone **Put Object** automation action, which should be used in more complex scenarios that involve additional logic or actions to be taken before or after uploading.

Prerequisites

This guide assumes you have completed the following prerequisites:

1. Created and configured an Amazon S3 bucket, IAM role and policy, and Amazon S3 connector in order to successfully integrate Amazon S3 with App Studio. To create a connector, you must have the Administrator role. For more information, see [Connect to Amazon Simple Storage Service \(Amazon S3\)](#).

Create an empty application

Create an empty application to use throughout this guide by performing the following steps.

To create an empty application

1. In the navigation pane, choose **My applications**.
2. Choose **+ Create app**.
3. In the **Create app** dialog box, give your application a name, choose **Build manually**, and choose **Next**.
4. In the **Select data sources** dialog box, choose **Skip** to create the application.

You will be navigated to the canvas of your new app, where you can use components, automations, and data to configure the look and function of your application.

Create pages

Create three pages in your application to gather or show information.

To create pages

1. If necessary, choose the **Pages** tab at the top of the canvas.
2. In the left-hand navigation, there is a single page that was created with your app. Choose **+ Add** twice to create two more pages. The navigation pane should show three total pages.

3. Update the name of the **Page1** page by performing the following steps:
 - a. Choose the ellipses icon and choose **Page properties**.
 - b. In the right-hand **Properties** menu, choose the pencil icon to edit the name.
 - c. Enter **FileList** and press enter.
4. Repeat the previous steps to update the second and third pages as follows:
 - Rename **Page2** to **UploadFile**.
 - Rename **Page3** to **FailUpload**.

Now, your app should have three pages named **FileList**, **UploadFile**, and **FailUpload** which can be seen in the left-hand **Pages** panel.

Next, you will create and configure the automations that interact with Amazon S3.

Create and configure automations

Create the automations of your application that interact with Amazon S3. Use the following procedures to create the following automations:

- A **getFiles** automation that lists the objects in your Amazon S3 bucket, which will be used to fill a table component.
- A **deleteFile** automation that deletes an object from your Amazon S3 bucket, which will be used to add a delete button to a table component.
- A **viewFile** automation that gets an object from your Amazon S3 bucket and displays it, which will be used to show more details about a single object selected from a table component.


Create a **getFiles** automation

Create an automation that will list the files in a specified Amazon S3 bucket.

1. Choose the **Automations** tab at the top of the canvas.
2. Choose **+ Add automation**.
3. In the right-hand panel, choose **Properties**.
4. Update the automation name by choosing the pencil icon. Enter **getFiles** and press enter.
5. Add a **List objects** action by performing the following steps:

- a. In the right-hand panel, choose **Actions**
 - b. Choose **List objects** to add an action. The action should be named `ListObjects1`.
6. Configure the action by performing the following steps:
- a. Choose the action from the canvas to open the right-hand **Properties** menu.
 - b. In **Connector**, choose the Amazon S3 connector that you created from the prerequisites.
 - c. In **Configuration**, enter the following text, replacing *bucket_name* with the bucket you created in the prerequisites:

```
{
  "Bucket": "bucket_name",
  "Prefix": ""
}
```

 **Note**

The `Prefix` field can be used to limit the response to objects that begin with the specified string.

7. The output of this automation will be used to populate a table component with objects from your Amazon S3 bucket, however, by default automations do not create outputs. Configure the automation to create an automation output by performing the following steps:
- a. In the left-hand navigation, choose the **getFiles** automation.
 - b. In the right-hand **Properties** menu, in **Automation output**, choose **+ Add output**.
 - c. In **Output**, enter `{{results.ListObjects1.Contents}}`. This expression returns the contents of the action, and can now be used to populate a table component.

Create a `deleteFile` automation

Create an automation that deletes an object from a specified Amazon S3 bucket.

1. In the left-hand **Automations** panel, choose **+ Add**.
2. Choose **+ Add automation**.
3. In the right-hand panel, choose **Properties**.
4. Update the automation name by choosing the pencil icon. Enter **deleteFile** and press enter.

5. Add an automation parameter, used to pass data to an automation, by performing the following steps:
 - a. In the right-hand **Properties** menu, in **Automation parameters**, choose **+ Add**.
 - b. Choose the pencil icon to edit the automation parameter. Update the parameter name to **fileName** and press enter.
6. Add a **Delete object** action by performing the following steps:
 - a. In the right-hand panel, choose **Actions**
 - b. Choose **Delete object** to add an action. The action should be named `DeleteObject1`.
7. Configure the action by performing the following steps:
 - a. Choose the action from the canvas to open the right-hand **Properties** menu.
 - b. In **Connector**, choose the Amazon S3 connector that you created from the prerequisites.
 - c. In **Configuration**, enter the following text, replacing *bucket_name* with the bucket you created in the prerequisites:

```
{
  "Bucket": "bucket_name",
  "Key": params.fileName
}
```

Create a `viewFile` automation

Create an automation that retrieves a single object from a specified Amazon S3 bucket. Later, you will configure this automation with a file viewer component to display the object.

1. In the left-hand **Automations** panel, choose **+ Add**.
2. Choose **+ Add automation**.
3. In the right-hand panel, choose **Properties**.
4. Update the automation name by choosing the pencil icon. Enter **viewFile** and press enter.
5. Add an automation parameter, used to pass data to an automation, by performing the following steps:
 - a. In the right-hand **Properties** menu, in **Automation parameters**, choose **+ Add**.

- b. Choose the pencil icon to edit the automation parameter. Update the parameter name to **fileName** and press enter.
6. Add a **Get object** action by performing the following steps:
 - a. In the right-hand panel, choose **Actions**
 - b. Choose **Get object** to add an action. The action should be named GetObject1.
7. Configure the action by performing the following steps:
 - a. Choose the action from the canvas to open the right-hand **Properties** menu.
 - b. In **Connector**, choose the Amazon S3 connector that you created from the prerequisites.
 - c. In **Configuration**, enter the following text, replacing *bucket_name* with the bucket you created in the prerequisites:

```
{
  "Bucket": "bucket_name",
  "Key": params.filename
}
```
8. By default, automations do not create outputs. Configure the automation to create an automation output by performing the following steps:
 - a. In the left-hand navigation, choose the **viewFile** automation.
 - b. In the right-hand **Properties** menu, in **Automation output**, choose **+ Add output**.
 - c. In **Output**, enter `{{results.GetObject1.Body.transformToWebStream()}}`. This expression returns the contents of the action.

Next, you will add components to the pages you created earlier, and configure them with your automations so users can use your app to view and delete files.

Add and configure page components

Now that you have created the automations that define the business logic and functionality of your app, you will create components and connect them both.

Add components to the FileList page

The **FileList** page that you created earlier will be used to display a list of files in the configured Amazon S3 bucket and more details about any file that is chosen from the list. To do that, you will do the following:

1. Create a table component to display the list of files. You will configure the table's rows to be filled with the output of the **getFiles** automation you previously created.
 2. Create a PDF viewer component to display a single PDF. You will configure the component to view a file selected from the table, using the **viewFile** automation you previously created to fetch the file from your bucket.
-
1. Choose the **Pages** tab at the top of the canvas.
 2. In the left-hand **Pages** panel, choose the **FileList** page.
 3. In the right-hand **Components** page, find the **Table** component and drag it to the center of the canvas.
 4. Choose the table component you just added to the page.
 5. In the right-hand **Properties** menu, choose the **Source** dropdown and select **Automation**.
 6. Choose the **Automation** dropdown and select the **getFiles** automation. The table will use the output of the **getFiles** automation as its content.
 7. Add a column to be filled with the name of the file.
 - a. In the right-hand **Properties** menu, next to **Columns**, choose **+ Add**.
 - b. Choose the arrow icon to the right of the **Column1** column that was just added.
 - c. In **Column label**, rename the column to **Filename**.
 - d. In **Value**, enter **{{currentRow.Key}}**.
 - e. Choose the arrow icon at the top of the panel to return to the main **Properties** panel.
 8. Add a table action to delete the file in a row.
 - a. In the right-hand **Properties** menu, next to **Actions**, choose **+ Add**.
 - b. In **Actions**, rename **Button** to **Delete**.
 - c. Choose the arrow icon to the right of the **Delete** action that was just renamed.
 - d. In **On click**, choose **+ Add action** and choose **Invoke automation**.
 - e. Choose the action that was added to configure it.

- f. In **Action name**, enter **DeleteRecord**.
 - g. In **Invoke automation**, select **deleteFile**.
 - h. In the parameter text box, enter **{{currentRow.Key}}**.
 - i. In **Value**, enter **{{currentRow.Key}}**.
9. In the right-hand panel, choose **Components** to view the components menu. There are two choices for showing files:
- An **Image viewer** to view files with a .png, .jpeg, or .jpg extension.
 - A **PDF viewer** component to view PDF files.

In this tutorial, you will add and configure the **PDF viewer** component.

10. Add the **PDF viewer** component.
- a. In the right-hand **Components** page, find the **PDF viewer** component and drag it to the canvas, below the table component.
 - b. Choose the **PDF viewer** component that was just added.
 - c. In the right-hand **Properties** menu, choose the **Source** dropdown and select **Automation**.
 - d. Choose the **Automation** dropdown and select the **viewFile** automation. The table will use the output of the **viewFile** automation as its content.
 - e. In the parameter text box, enter **{{ui.table1.selectedRow["Filename"]}}**.
 - f. In the right-hand panel, there is also a **File name** field. The value of this field is used as the header for the PDF viewer component. Enter the same text as the previous step: **{{ui.table1.selectedRow["Filename"]}}**.

Add components to the UploadFile page

The **UploadFile** page will contain a file selector that can be used to select and upload a file to the configured Amazon S3 bucket. You will add the **S3 upload** component to the page, which users can use to select and upload a file.

1. In the left-hand **Pages** panel, choose the **UploadFile** page.
2. In the right-hand **Components** page, find the **S3 upload** component and drag it to the center of the canvas.
3. Choose the S3 upload component you just added to the page.

4. In the right-hand **Properties** menu, configure the component:
 - a. In the **Connector** dropdown, select the Amazon S3 connector that was created in the prerequisites.
 - b. In the **Bucket** text box, enter the name of your Amazon S3 bucket.
 - c. In the **File name** text box, enter `{{ui.s3Upload1.files[0]?.nameWithExtension}}`.
 - d. In the **Max file size**, enter **5** in the textbox and ensure **MB** is selected in the dropdown.
 - e. In **Triggers** section, add actions that run after successful or unsuccessful uploads by performing the following steps:

To add an action that runs after successful uploads:

1. In **On success**, choose **+ Add action** and select **Navigate**.
2. Choose the action that was added to configure it.
3. In **Navigation type**, choose **Page**.
4. In **Navigate to**, choose **FailUpload**.
5. Choose the arrow icon at the top of the panel to return to the main **Properties** panel.

To add an action that runs after unsuccessful uploads:

1. In **On success**, choose **+ Add action** and select **Navigate**.
2. Choose the action that was added to configure it.
3. In **Navigation type**, choose **Page**.
4. In **Navigate to**, choose **FileList**.
5. Choose the arrow icon at the top of the panel to return to the main **Properties** panel.

Add components to the FailUpload page

The **FailUpload** page is a simple page containing a text box that informs users that their upload failed.

1. In the left-hand **Pages** panel, choose the **FailUpload** page.
2. In the right-hand **Components** page, find the **Text** component and drag it to the center of the **canvas**.

3. Choose the text component you just added to the page.
4. In the right-hand **Properties** menu, in **Value**, enter **Failed to upload, try again**.

Next steps: Preview and publish the application for testing

The application is now ready for testing. For more information about previewing and publishing applications, see [Previewing, publishing, and sharing applications](#).

Creating, editing, and deleting automations

Contents

- [Creating an automation](#)
- [Viewing or editing automation properties](#)
- [Deleting an automation](#)

Creating an automation

Use the following procedure to create an automation in an App Studio application. Once created, an automation must be configured by editing its properties and adding actions to it.

To create an automation

1. If necessary, navigate to the application studio of your application.
2. Choose the **Automations** tab.
3. If you have no automations, choose **+ Add automation** in the canvas. Otherwise, in the left-side **Automations** menu, choose **+ Add**.
4. A new automation will be created, and you can start editing its properties or adding and configuring actions to define your application's business logic.

Viewing or editing automation properties

Use the following procedure to view or edit automation properties.

To view or edit automation properties

1. If necessary, navigate to the application studio of your application.

2. Choose the **Automations** tab.
3. In the left-hand **Automations** menu, choose the automation for which you want to view or edit properties to open the right-side **Properties** menu.
4. In the **Properties** menu, you can view the following properties:
 - **Automation identifier:** The unique name of the automation. To edit it, enter a new identifier in the text field.
 - **Automation parameters:** Automation parameters are used to pass dynamic values from your app's UI to automation and data actions. To add a parameter, choose **+ Add**. Choose the pencil icon to change the parameter's name, description, or type. To remove a parameter, choose the trash icon.

 **Tip**

You can also add automation parameters directly from the canvas.

- **Automation output:** The automation output is used to configure which data from the automation can be referenced in other automations or components. By default, automations do not create an output. To add an automation output choose **+ Add**. To remove the output, choose the trash icon.
5. You define what an automation does by adding and configuring actions. For more information about actions, see [Adding, editing, and deleting automation actions](#) and [Automation actions reference](#).

Deleting an automation

Use the following procedure to delete an automation in an App Studio application.

To delete an automation

1. If necessary, navigate to the application studio of your application.
2. Choose the **Automations** tab.
3. In the left-side **Automations** menu, choose the ellipses menu of the automation you want to delete, and choose **Delete**. Alternatively, you can choose the trash icon from the right-side **Properties** menu of the automation.
4. In the confirmation dialog box, choose **Delete**.

Adding, editing, and deleting automation actions

An automation action, commonly referred to as an **action**, is an individual step of logic that make up an automation. Each action performs a specific task, whether it's sending an email, creating a data record, invoking a Lambda function, or calling APIs. Actions are added to automations from the action library, and can be grouped into conditional statements or loops.

Contents

- [Adding an automation action](#)
- [Viewing and editing automation action properties](#)
- [Deleting an automation action](#)

Adding an automation action

Use the following procedure to add an action to an automation in an App Studio application.

To add an automation action

1. If necessary, navigate to the application studio of your application.
2. Choose the **Automations** tab.
3. In the left-side **Automations** menu, choose the automation you want to add an action to.
4. In the right-hand **Action** menu, choose the action you want to add, or drag and drop the action into the canvas. After the action is created, you can choose the action to configure the action properties to define the action's functionality. For more information about action properties and configuring them, see [Automation actions reference](#).

Viewing and editing automation action properties

Use the following procedure to view or edit an automation action's properties in an App Studio application.

To view or edit automation action properties

1. If necessary, navigate to the application studio of your application.
2. Choose the **Automations** tab.

3. In the left-side **Automations** menu, choose the action of which you want to view or edit properties. Alternatively, you can choose the action in the canvas when viewing the automation that contains it.
4. You can view or edit the action properties in the right-side **Properties** menu. The properties for an action are different for each action type. For more information about action properties and configuring them, see [Automation actions reference](#).

Deleting an automation action

Use the following procedure to delete an action from an automation in an App Studio application.

To delete an automation action

1. If necessary, navigate to the application studio of your application.
2. Choose the **Automations** tab.
3. In the left-side **Automations** menu, choose the automation that contains the action you want to delete.
4. In the canvas, choose the trash icon in the action you want to delete and choose **Delete**.

Automation actions reference

The following is the reference documentation for automation actions used in App Studio.

An automation action, commonly referred to as an **action**, is an individual step of logic that make up an automation. Each action performs a specific task, whether it's sending an email, creating a data record, invoking a Lambda function, or calling APIs. Actions are added to automations from the action library, and can be grouped into conditional statements or loops.

For information about creating and configuring automations and their actions, see the topics in [Defining and implementing your app's business logic with automations](#).

Invoke API

Invokes an HTTP REST API request. Builders can use this action to send requests from App Studio to other systems or services with APIs. For example, you could use it to connect to third-party systems or homegrown applications to access business critical data, or invoke API endpoints that cannot be called by dedicated App Studio actions.

Properties

Connector

The **API connector** to use for the API requests made by this action. The connector dropdown will be filtered to show only API-related connectors. Depending on how the connector is configured, it can contain important information such as credentials and default headers or query parameters. For more information about API connectors, see [Connect to third-party services](#).

API request configuration properties

Choose **Configure API request** from the properties panel to open the request configuration dialog box. If an **API connector** is selected, the dialog box will include connector information.

Method: The method for the API call. Possible values are as follows:

- DELETE: Deletes a specified resource.
- GET: Retrieves information or data.
- HEAD: Retrieves only the headers of a response without the body.
- POST: Submits data to be processed.
- PUSH: Submits data to be processed.
- PATCH: Partially updates a specified resource.

Path: The relative path to the resource.

Headers: Any headers in the form of key-value pairs to be sent with the API request. If a connector is selected, its configured headers will be automatically added and cannot be removed. The configured headers cannot be edited, but you can override them by adding another header with the same name.

Query parameters: Any query parameters in the form of key-value pairs to be sent with the API request. If a connector is selected, its configured query parameters will be automatically added and cannot be edited or removed.

Body: Information to be sent with the API request in JSON format. There is no body for GET requests.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Invoke AWS

Invokes an operation from an AWS service. This is a general action for calling AWS services or operations, and should be used if there is not a dedicated action for the desired AWS service or operation.

Properties

Service

The AWS service which contains the operation to be run.

Operation

The operation to be run.

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The JSON input to be when running the specified operation. For more information about configuring inputs for AWS operations, see the [AWS SDK for JavaScript](#).

Invoke Lambda

Invokes an existing Lambda function.

Properties

Connector

The connector to be used for the Lambda functions run by this action. The configured connector should be set up with the proper credentials to access the Lambda function, and other configuration information, such as the AWS region that contains the Lambda function. For more information about configuring a connector for Lambda, see [Create Lambda connector](#).

Function name

The name of the Lambda function to be run. Note that this is the function name, and not the function ARN (Amazon Resource Name).

Function event

Key-value pairs to be passed along to your Lambda function as the event payload.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Loop

Runs nested actions repeatedly to iterate through a list of items, one item at a time. For example, add the [Create record](#) action to a loop action to create multiple records.

The loop action can be nested within other loops or condition actions. The loop actions are run sequentially, and not in parallel. The results of each action within the loop can only be accessed to subsequent actions within the same loop iteration. They cannot be accessed outside of the loop or in different iterations of the loop.

Properties

Source

The list of items to iterate through, one item at a time. The source can be the result of a previous action or a static list of strings, numbers, or objects that you can provide using a JavaScript expression.

Examples

The following list contains examples of source inputs.

- Results from a previous action: `{{results.actionName.data}}`
- A list of numbers: `{{[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}}`
- A list of strings: `{{["apple", "banana", "orange", "grape", "kiwi"]}}`
- A computed value: `{{params.actionName.split("\n")}}`

Current item name

The name of the variable that can be used to reference the current item being iterated. The current item name is configurable so that you can nest two or more loops and access variables from each loop. For example, if you are looping through countries and cities with two loops, you could configure and reference `currentCountry` and `currentCity`.

Condition

Runs actions based on the result of one or more specified logical conditions that are evaluated when the automation is run. The condition action is made up of the following components:

- A *condition* field, which is used to provide a JavaScript expression that evaluates to `true` or `false`.
- A *true branch*, which contains actions that are run if the condition evaluates to `true`.
- A *false branch*, which contains actions that are run if the condition evaluates to `false`.

Add actions to the true and false branches by dragging them into the condition action.

Properties

Condition

The JavaScript expression to be evaluated when the action is run.

Create record

Creates one record in an existing App Studio entity.

Properties

Entity

The entity in which a record is to be created. Once an entity is selected, values must be added to the entity's fields for the record to be created. The types of the fields, and if the fields are required or optional are defined in the entity.

Update record

Updates an existing record in an App Studio entity.

Properties

Entity

The entity that contains the records to be updated.

Conditions

The criteria that defines which records are updated by the action. You can group conditions to create one logical statement. You can combine groups or conditions with AND or OR statements.

Fields

The fields to be updated in the records specified by the conditions.

Values

The values to be updated in the specified fields.

Delete record

Deletes a record from an App Studio entity.

Properties

Entity

The entity that contains the records to be deleted.

Conditions

The criteria that defines which records are deleted by the action. You can group conditions to create one logic statement. You can combine groups or conditions with AND or OR statements.

Invoke data action

Runs a data action with optional parameters.

Properties

Data action

The data action to be run by the action.

Parameters

Data action parameters to be used by the data action. Data action parameters are used to send values that are used as inputs for data actions. Data action parameters can be added when configuring the automation action, but must be edited in the **Data** tab.

Advanced settings

The `Invoke data action` action contains the following advanced settings:

- **Page size:** The maximum number of records to fetch in each query. The default value is 500, and the maximum value is 3000.
- **Pagination token:** The token used to fetch additional records from a query. For example, if the `Page size` is set to 500, but there are more than 500 records, passing the pagination token to a subsequent query will fetch the next 500. The token will be undefined if no more records or pages exist.

Amazon S3: Put object

Uses the Amazon `S3 PutObject` operation to add an object identified by a key (file path) to a specified Amazon S3 bucket.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the appropriate credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The required options to be used in the PutObject command. The options are as follows:

Note

For more information about the Amazon S3 PutObject operation, see [PutObject](#) in the *Amazon Simple Storage Service API Reference*.

- **Bucket:** The name of the Amazon S3 bucket in which to put an object.
- **Key:** The unique name of the object to be put into the Amazon S3 bucket.
- **Body:** The content of the object to be put into the Amazon S3 bucket.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon S3: Delete object

Uses the Amazon S3 DeleteObject operation to delete an object identified by a key (file path) from a specified Amazon S3 bucket.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The required options to be used in the `DeleteObject` command. The options are as follows:

Note

For more information about the Amazon S3 `DeleteObject` operation, see [DeleteObject](#) in the *Amazon Simple Storage Service API Reference*.

- **Bucket:** The name of the Amazon S3 bucket from which to delete an object.
- **Key:** The unique name of the object to be deleted from the Amazon S3 bucket.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon S3: Get object

Uses the Amazon S3 `GetObject` operation to retrieve an object identified by a key (file path) from a specified Amazon S3 bucket.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The required options to be used in the `GetObject` command. The options are as follows:

Note

For more information about the Amazon S3 `GetObject` operation, see [GetObject](#) in the *Amazon Simple Storage Service API Reference*.

- **Bucket:** The name of the Amazon S3 bucket from which to retrieve an object.
- **Key:** The unique name of the object to be retrieved from the Amazon S3 bucket.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon S3: List objects

Uses the Amazon S3 `ListObjects` operation to list objects in a specified Amazon S3 bucket.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The required options to be used in the `ListObjects` command. The options are as follows:

Note

For more information about the Amazon S3 `ListObjects` operation, see [ListObjects](#) in the *Amazon Simple Storage Service API Reference*.

- **Bucket:** The name of the Amazon S3 bucket from which to list objects.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon Textract: Analyze document

Uses the Amazon Textract `AnalyzeDocument` operation to analyze an input document for relationships between detected items.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The content of the request to be used in the `AnalyzeDocument` command. The options are as follows:

Note

For more information about the Amazon Textract `AnalyzeDocument` operation, see [AnalyzeDocument](#) in the *Amazon Textract Developer Guide*.

- **Document / S3Object / Bucket:** The name of the Amazon S3 bucket. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Name:** The file name of the input document. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Version:** If the Amazon S3 bucket has versioning enabled, you can specify the version of the object. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **FeatureTypes:** A list of the types of analysis to perform. Valid values are: TABLES, FORMS, QUERIES, SIGNATURES, and LAYOUT.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon Textract: Analyze expense

Uses the Amazon Textract `AnalyzeExpense` operation to analyze an input document for financially-related relationships between text.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The content of the request to be used in the `AnalyzeExpense` command. The options are as follows:

Note

For more information about the Amazon Textract `AnalyzeExpense` operation, see [AnalyzeExpense](#) in the *Amazon Textract Developer Guide*.

- **Document / S3Object / Bucket:** The name of the Amazon S3 bucket. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Name:** The file name of the input document. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Version:** If the Amazon S3 bucket has versioning enabled, you can specify the version of the object. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon Textract: Analyze ID

Uses the Amazon Textract `AnalyzeID` operation to analyze an identity document for relevant information.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The content of the request to be used in the `AnalyzeID` command. The options are as follows:

Note

For more information about the Amazon Textract `AnalyzeID` operation, see [AnalyzeID](#) in the *Amazon Textract Developer Guide*.

- **Document / S3Object / Bucket:** The name of the Amazon S3 bucket. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Name:** The file name of the input document. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Version:** If the Amazon S3 bucket has versioning enabled, you can specify the version of the object. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the

preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon Textract: Detect doc text

Uses the Amazon Textract `DetectDocumentText` operation to detect lines of text and the words that make up a line of text in an input document.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The content of the request to be used in the `DetectDocumentText` command. The options are as follows:

Note

For more information about the Amazon Textract `DetectDocumentText` operation, see [DetectDocumentText](#) in the *Amazon Textract Developer Guide*.

- **Document / S3Object / Bucket:** The name of the Amazon S3 bucket. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Name:** The file name of the input document. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.
- **Document / S3Object / Version:** If the Amazon S3 bucket has versioning enabled, you can specify the version of the object. This parameter can be left empty if a file is passed to the action with the **S3 upload** component.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Amazon Bedrock: Invoke model

Uses the [Amazon Bedrock InvokeModel](#) operation to run inference using the prompt and inference parameters provided in the request body. You use model inference to generate text, images, and embeddings.

Properties

Connector

The connector to be used for the operations run by this action. To use this action successfully, the connector must be configured with **Amazon Bedrock Runtime** as the service. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The content of the request to be used in the `InvokeModel` command.

Note

For more information about the Amazon Bedrock `InvokeModel` operation, including example commands, see [InvokeModel](#) in the *Amazon Bedrock API Reference*.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the

preview environment for testing purposes. This snippet is stored in the action's `results` map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

JavaScript

Runs a custom JavaScript function to return a specified value.

Properties

Source code

The JavaScript code snippet to be run by the action.

Invoke automation

Runs a specified automation.

Properties

Invoke Automation

The automation to be run by the action.

Send email

Uses the Amazon SES `SendEmail` operation to send an email.

Properties

Connector

The connector to be used for the operations run by this action. The configured connector should be set up with the proper credentials to run the operation, and other configuration information, such as the AWS region that contains any resources referenced in the operation.

Configuration

The content of the request to be used in the `SendEmail` command. The options are as follows:

Note

For more information about the Amazon SES SendEmail operation, see [SendEmail](#) in the *Amazon Simple Email Service API Reference*.

Mocked output

Actions do not interact with external services or resources in the preview environment. The **Mocked output** field is used to provide a JSON expression that simulates the behavior of a connector in the preview environment for testing purposes. This snippet is stored in the action's results map, just like the connector response would be for a published app in the live environment.

With this field, you can test various scenarios and their impact on other actions within the automation such as simulating different result values, error scenarios, edge cases, or unhappy paths without communicating with external services through connectors.

Configure your app's data model with entities

Entities are data tables in App Studio. Entities interact directly with tables in data sources. Entities include fields to describe the data in them, queries to locate and return data, and mapping to connect the entity's fields to a data source's columns.

Topics

- [Creating an entity in an App Studio app](#)
- [Configuring or editing an entity in an App Studio app](#)
- [Deleting an entity](#)
- [Managed data entities in AWS App Studio](#)

Creating an entity in an App Studio app

There are four methods for creating an entity in an App Studio app. The following list contains each method, its benefits, and a link to the instructions for using that method to create and then configure the entity.

- [Creating an entity from an existing data source](#): Automatically create an entity and its fields from an existing data source table and map the fields to the data source table columns. This option is preferable if you have an existing data source that you want to use in your App Studio app.
- [Creating an entity with an App Studio managed data source](#): Create an entity and a DynamoDB table that App Studio manages for you. The DynamoDB table is automatically updated as you update your entity. With this option, you don't have to manually create, manage, or connect a third-party data source, or designate mapping from entity fields to table columns. All of your app's data modeling and configuration is done in App Studio. This option is preferable if you don't want to manage your own data sources and a DynamoDB table and its functionality is sufficient for your app.
- [Creating an empty entity](#): Create an empty entity entirely from scratch. This option is preferable if you don't have any existing data sources or connectors created by an admin, and you want to flexibly design your app's data model without being constrained by external data sources. You can connect the entity to a data source after creation.
- [Creating an entity with AI](#): Generate an entity, fields, data actions, and sample data based on the specified entity name. This option is preferable if you have an idea of the data model for your app, but you want help translating it into an entity.

Creating an entity from an existing data source

Use a table from a data source to automatically create an entity and its fields, and map the entity fields to the columns of the table. This option is preferable if you have an existing data source that you want to use in your App Studio app.

1. If necessary, navigate to your application.
2. Choose the **Data** tab at the top of the canvas.
3. If there are no entities in your app, choose **+ Create entity**. Otherwise, in the left-side **Entities** menu, choose **+ Add**.
4. Select **Use a table from an existing data source**.
5. In **Connector**, select the connector that contains the table you want to use to create your entity.
6. In **Table**, choose the table you want to use to create your entity.
7. Select the **Create data actions** checkbox to create data actions.
8. Choose **Create entity**. Your entity is now created, and you can see it in the left-hand **Entities** panel.

9. Configure your new entity by following the procedures in [Configuring or editing an entity in an App Studio app](#). Note that because your entity was created with an existing data source, some properties or resources have already been created, such as fields, the connected data source, and field mapping. Also, your entity will contain data actions if you selected the **Create data actions** checkbox during creation.

Creating an entity with an App Studio managed data source

Create a managed entity and corresponding DynamoDB table that is managed by App Studio. While the DynamoDB table exists in the associated AWS account, when changes are made to the entity in the App Studio app, the DynamoDB table is updated automatically. With this option, you don't have to manually create, manage, or connect a third-party data source, or designate mapping from entity fields to table columns. This option is preferable if you don't want to manage your own data sources and a DynamoDB table and its functionality is sufficient for your app. For more information about managed entities, see [Managed data entities in AWS App Studio](#).

You can use the same managed entities in multiple applications. For instructions, see [Creating an entity from an existing data source](#).

1. If necessary, navigate to your application.
2. Choose the **Data** tab at the top of the canvas.
3. If there are no entities in your app, choose **+ Create entity**. Otherwise, in the left-side **Entities** menu, choose **+ Add**.
4. Select **Create App Studio managed entity**.
5. In **Entity name**, provide a name for your entity.
6. In **Primary key**, provide a name for the primary key of your entity. The primary key is the unique identifier of the entity and cannot be changed after the entity is created.
7. In **Primary key data type**, select the data type of primary key of your entity. The data type cannot be changed after the entity is created.
8. Choose **Create entity**. Your entity is now created, and you can see it in the left-hand **Entities** panel.
9. Configure your new entity by following the procedures in [Configuring or editing an entity in an App Studio app](#). Note that because your entity was created with managed data, some properties or resources have already been created, such as the primary key field, and the connected data source.

Creating an empty entity

Create an empty entity entirely from scratch. This option is preferable if you don't have any existing data sources or connectors created by an admin. Creating an empty entity offers flexibility, as you can design your entity within your App Studio app without being constrained by external data sources. After you design your app's data model, and configure the entity accordingly, you can still connect it to an external data source later.

1. If necessary, navigate to your application.
2. Choose the **Data** tab at the top of the canvas.
3. If there are no entities in your app, choose **+ Create entity**. Otherwise, in the left-side **Entities** menu, choose **+ Add**.
4. Select **Create an entity**.
5. Choose **Create entity**. Your entity is now created, and you can see it in the left-hand **Entities** panel.
6. Configure your new entity by following the procedures in [Configuring or editing an entity in an App Studio app](#).

Creating an entity with AI

Generate an entity, fields, data actions, and sample data based on the specified entity name. This option is preferable if you have an idea of the data model for your app, but you want help translating it into an entity.

1. If necessary, navigate to your application.
2. Choose the **Data** tab at the top of the canvas.
3. If there are no entities in your app, choose **+ Create entity**. Otherwise, in the left-side **Entities** menu, choose **+ Add**.
4. Select **Create an entity with AI**.
5. In **Entity name**, provide a name for your entity. This name is used to generate the fields, data actions, and sample data of your entity.
6. Select the **Create data actions** checkbox to create data actions.
7. Choose **Generate an entity**. Your entity is now created, and you can see it in the left-hand **Entities** panel.

8. Configure your new entity by following the procedures in [Configuring or editing an entity in an App Studio app](#). Note that because your entity was created with AI, your entity will already contain generated fields. Also, your entity will contain data actions if you selected the **Create data actions** checkbox during creation.

Configuring or editing an entity in an App Studio app

Use the following topics to configure an entity in an App Studio application.

Topics

- [Editing the entity name](#)
- [Adding, editing, or deleting entity fields](#)
- [Creating, editing, or deleting data actions](#)
- [Adding or deleting sample data](#)
- [Add or edit connected data source and map fields](#)

Editing the entity name

1. If necessary, navigate to the entity you want to edit.
2. In the **Configuration** tab, in **Entity name**, update the entity name and choose outside of the text box to save your changes.

Adding, editing, or deleting entity fields

Tip

You can press CTRL+Z to undo the most recent change to your entity.

1. If necessary, navigate to the entity you want to edit.
2. In the **Configuration** tab, in **Fields**, you view a table of your entity's fields. Entity fields have the following columns:

- **Display name:** The display name is similar to a table header or form field and is viewable by application users. It can contain spaces and special characters but must be unique within an entity.
 - **System name:** The system name is a unique identifier used in code to reference a field. When mapping to a column in an Amazon Redshift table, it must match the Amazon Redshift table column name.
 - **Data type:** The type of data that will be stored within this field, such as Integer, Boolean, or String.
3. To add fields:
 - a. To use AI to generate fields based on entity name and connected data source, choose **Generate more fields**.
 - b. To add a single field, choose **+ Add field**.
 4. To edit a field:
 - a. To edit the display name, enter the desired value in the **Display name** text box. If the system name of the field hasn't been edited, it will be updated to the new value of the display name.
 - b. To edit the system name, enter the desired value in the **System name** text box.
 - c. To edit the data type, choose the **Data type** dropdown menu and select the desired type from the list.
 - d. To edit the field's properties, choose the gear icon of the field. The following list details the field properties:
 - **Required:** Enable this option if the field is required by your data source.
 - **Primary key:** Enable this option if the field is mapped to a primary key in your data source.
 - **Unique:** Enable this option if the value of this field must be unique.
 - **Use data source default:** Enable this option if the value of the field is provided by the data source, such as using auto-increment, or an event timestamp.
 - **Data type options:** Fields of certain data types can be configured with data type options such as minimum or maximum values.
 5. To delete a field, choose the trash icon of the field you want to delete.

Creating, editing, or deleting data actions

Data actions are used in applications to run actions on an entity's data, such as fetching all records, or fetching a record by ID. Data actions can be used to locate and return data matching specified conditions to be viewed in components such as tables or detail views.

Contents

- [Creating data actions](#)
- [Editing or configuring data actions](#)
- [Deleting data actions](#)

Creating data actions

Tip

You can press CTRL+Z to undo the most recent change to your entity.

1. If necessary, navigate to the entity for which you want to create data actions.
2. Choose the **Data actions** tab.
3. There are two methods for creating data actions:
 - (Recommended) To use AI to generate data actions for you, based on your entity name, fields, and connected data source, choose **Generate data actions**. The following actions will be generated:
 1. `getAll`: Retrieves all the records from an entity. This action is useful when you need to display a list of records or perform operations on multiple records at once.
 2. `getByID`: Retrieves a single record from an entity based on its unique identifier (ID or primary key). This action is useful when you need to display or perform operations on a specific record.
 - To add a single data action, choose **+ Add data action**.
4. To view or configure the new data action, see the following section, [Editing or configuring data actions](#).

Editing or configuring data actions

1. If necessary, navigate to the entity for which you want to create data actions.
2. Choose the **Data actions** tab.
3. In **Fields** configure the fields to be returned by the query. By default, all of the configured fields in the entity are selected.

You can also add **Joins** to the data action by performing the following steps:

1. Choose **+ Add Join** to open a dialog box.
2. In **Related entity**, select the entity you want to join with the current entity.
3. In **Alias**, optionally enter a temporary alias name for the related entity.
4. In **Join type**, select the desired join type.
5. Define the join clause by selecting the fields from each entity.
6. Choose **Add** to create the join.

Once created, the join will be displayed in the **Joins** section, making additional fields available in the **Fields to Return** dropdown. You can add multiple joins, including chained joins across entities. You can also filter and sort by fields from joined entities.

To delete a join, choose the trash icon next to it. This will remove any fields from that join and break any dependent joins or constraints using those fields.

4. In **Conditions**, add, edit, or remove rules that filter the output of the query. You can organize rules into groups, and you can chain together multiple rules with AND or OR statements.
5. In **Sorting**, configure how the query results are sorted by choosing an attribute and choosing ascending or descending order. You can remove the sorting configuration by choosing the trash icon next to the sorting rule.
6. In **Transform results**, you can enter custom JavaScript to modify or format results before they are displayed or sent to automations.
7. In **Output preview**, view a preview table of the query output based on the configured fields, filters, sorting, and JavaScript.

Deleting data actions

Use the following procedure to delete data actions from an App Studio entity.

1. If necessary, navigate to the entity for which you want to delete data actions.
2. Choose the **Data actions** tab.
3. For each data action you want to delete, choose the dropdown menu next to **Edit** and choose **Delete**.
4. Choose **Confirm** in the dialog box.

Adding or deleting sample data

You can add sample data to entities in an App Studio application. Because applications don't communicate with external services until they are published, sample data can be used to test your application and entity in preview environments.

1. If necessary, navigate to the entity you want to edit.
2. Choose the **Sample data** tab.
3. To generate sample data, choose **Generate more sample data**.
4. To delete sample data, select the checkboxes of the data you want to delete, and press the Delete or Backspace key. Choose **Save** to save the changes.

Add or edit connected data source and map fields

Tip

You can press CTRL+Z to undo the most recent change to your entity.

1. If necessary, navigate to the entity you want to edit.
2. Choose the **Connection** tab to view or manage the connection between the entity and a data source table where data is stored when your application is published. Once a data source table is connected, you can map the entity fields to the columns of the table.
3. In **Connector**, choose the connector that contains a connection to the desired data source table. For more information about connectors, see [Connect App Studio to other services with connectors](#).
4. In **Table**, choose the table you want to use as a data source for the entity.
5. The table shows the fields of entity, and the data source column they are mapped to. Choose **Auto map** to automatically map your entity fields with your data source columns. You can also

map fields manually in the table by choosing the data source column in the dropdown for each entity field.

Deleting an entity

Use the following procedure to delete an entity from an App Studio application.

Note

Deleting an entity from an App Studio app does not delete the connected data source table, including the corresponding DynamoDB table of managed entities. The data source tables will remain in the associated AWS account and will need to be deleted from the corresponding service if desired.

To delete an entity

1. If necessary, navigate to your application.
2. Choose the **Data** tab.
3. In the left-hand **Entities** menu, choose the ellipses menu next to the entity you want to delete and choose **Delete**.
4. Review the information in the dialog box, enter **confirm** and choose **Delete** to delete the entity.

Managed data entities in AWS App Studio

Typically, you configure an entity in App Studio with a connection to an external database table, and you must create and map each entity field with a column in the connected database table. When you make a change to the data model, both the external database table and the entity must be updated, and the changed fields must be remapped. While this method is flexible and enables the use of different types of data sources, it takes more up-front planning and ongoing maintenance.

A *managed entity* is a type of entity for which App Studio manages the entire data storage and configuration process for you. When you create a managed entity, a corresponding DynamoDB table is created in the associated AWS account. This ensures secure and transparent data

management within AWS. With a managed entity, you configure the entity's schema in App Studio, and the corresponding DynamoDB table is automatically updated as well.

Using managed entities in multiple applications

Once you create a managed entity in an App Studio app, that entity can be used in other App Studio apps. This is helpful for configuring data storage for apps with identical data models and schemas by providing a single underlying resource to maintain.

When using a managed entity in multiple applications, all schema updates to the corresponding DynamoDB table must be made using the original application in which the managed entity was created. Any schema changes made to the entity in other applications will not update the corresponding DynamoDB table.

Managed entity limitations

Primary key update restrictions: You cannot change the entity's primary key name or type after it is created, as this is a destructive change in DynamoDB, and would result in loss of existing data.

Renaming columns: When you rename a column in DynamoDB, you actually create a new column while the original column remains with original data. The original data is not automatically copied to the new column or deleted from the original column. You can rename managed entity fields, known as the *system name*, but you will lose access to the original column and its data. There is no restriction with renaming the display name.

Changing data type: Though DynamoDB allows flexibility to modify column data types after table creation, such changes can severely impact existing data as well as query logic and accuracy. Data type changes require transforming all existing data to conform to the new format, which is complex for large, active tables. Additionally, data actions may return unexpected results until data migration is complete. You can switch data types of fields, but the existing data will not be migrated to the new data type.

Sorting Column: DynamoDB enables sorted data retrieval through Sort Keys. Sort Keys must be defined as part of composite Primary Keys along with the Partition Key. Limitations include mandatory Sort Key, sorting confined within one partition, and no global sorting across partitions. Careful data modeling of Sort Keys is required to avoid hot partitions. We will not be supporting Sorting for Preview milestone.

Joins: Joins are not supported in DynamoDB. Tables are denormalized by design to avoid expensive join operations. To model one-to-many relationships, the child table contains an attribute

referencing the parent table's primary key. Multi-table data queries involve looking up items from the parent table to retrieve details. We will not be supporting native Joins for Managed entities as part of the Preview milestone. As a workaround, we will introduce an automation step that can perform a data merge of 2 entities. This will be very similar to a one level look-up. We will not be supporting Sorting for Preview milestone.

Env Stage: We will allow publishing to test but use the same managed store across both environments

Page and automation parameters

Parameters are a powerful feature in AWS App Studio that are used to pass dynamic values between different components, pages, and automations within your application. Using parameters, you can make flexible and context-aware experiences, making your applications more responsive and personalized. This article covers two types of parameters: page parameters and automation parameters.

Topics

- [Page parameters](#)
- [Automation parameters](#)

Page parameters

Page parameters are a way to send information between pages and are often used when navigating from one page to another within an App Studio app to maintain context or pass data. Page parameters typically consist of a name and a value.

Page parameter use cases

Page parameters are used for passing data between different pages and components within your App Studio applications. They are particularly helpful for the following use cases:

1. **Searching and filtering:** When users search on your app's homepage, the search terms can be passed as parameters to the results page, allowing it to display only the relevant filtered items. For example, if a user searches for *noise-cancelling headphones*, the parameter with the value *noise-cancelling headphones* can be passed to the product listing page.
2. **Viewing item details:** If a user clicks on a listing, such as a product, the unique identifier of that item can be passed as a parameter to the details page. This allows the details page to display

all the information about the specific item. For example, when a user clicks on a headphone product, the product's unique ID is passed as a parameter to the product details page.

3. **Passing user context in page navigation:** As users navigate between pages, parameters can pass along important context, such as the user's location, preferred product categories, shopping cart contents, and other settings. For example, as a user browses through different product categories on your app, their location and preferred categories are retained as parameters, providing a personalized and consistent experience.
4. **Deep links:** Use page parameters to share or bookmark a link to a specific page within the app.
5. **Data actions:** You can create data actions that accept parameter values to filter and query your data sources based on the passed parameters. For example, on the product listing page, you can create a data action that accepts category parameters to fetch the relevant products.

Page parameter security considerations

While page parameters provide a powerful way to pass data between pages, you must use them with caution, as they can potentially expose sensitive information if not used properly. Here is an important security considerations to keep in mind:

1. Avoid exposing sensitive data in URLs

- a. **Risk:** URLs, including data action parameters, are often visible in server logs, browser history, and other places. As such, it's essential to avoid exposing sensitive data, such as user credentials, personal identifiable information (PII), or any other confidential data, in page parameter values.
- b. **Mitigation:** Consider using identifiers that can be securely mapped to the sensitive data. For example, instead of passing a user's name or email address as a parameter, you could pass a random unique identifier that can be used to fetch the user's name or email.

Automation parameters

Automation parameters are a powerful feature in App Studio that can be used to create flexible and reusable automations by passing dynamic values from various sources, such as the UI, other automations, or data actions. They act as placeholders that are replaced with actual values when the automation is run, allowing you to use the same automation with different inputs each time.

Within an automation, parameters have unique names, and you can reference a parameter's value using the `params` variable followed by the parameter's name, for example, `{{params.customerId}}`.

This article provides an in-depth understanding of automation parameters, including their fundamental concepts, usage, and best practices.

Automation parameter benefits

Automation parameters provide several benefits, including the following list:

1. **Reusability:** By using parameters, you can create reusable automations that can be customized with different input values, allowing you to reuse the same automation logic with different inputs.
2. **Flexibility:** Instead of hard-coding values into an automation, you can define parameters and provide different values when needed, making your automations more dynamic and adaptable.
3. **Separation of concerns:** Parameters help separate the automation logic from the specific values used, promoting code organization and maintainability.
4. **Validation:** Each parameter has a data type, such as string, number, or boolean, which is validated at runtime. This ensures that requests with incorrect data types are rejected without the need for custom validation code.
5. **Optional and required parameters:** You can designate automation parameters as optional or required. Required parameters must be provided when running the automation, while optional parameters can have default values or be omitted. This flexibility allows you to create more versatile automations that can handle different scenarios based on the provided parameters.

Scenarios and use cases

Scenario: Retrieving product details

Imagine you have an automation that retrieves product details from a database based on a product ID. This automation could have a parameter called `productId`.

The `productId` parameter acts as a placeholder that you can fill in with the actual product ID value when running the automation. Instead of hard-coding a specific product ID into the automation, you can define the `productId` parameter and pass in different product ID values each time you run the automation.

You could call this automation from a component's data source, passing the selected product's ID as the `productId` parameter using the double curly bracket syntax: `{{ui.productsTable.selectedRow.id}}`. This way, when a user selects a product from a table (`ui.productsTable`), the automation will retrieve the details for the selected product by passing the id of the selected row as the `productId` parameter.

Alternatively, you could invoke this automation from another automation that loops over a list of products and retrieves the details for each product by passing the product's id as the `productId` parameter. In this scenario, the `productId` parameter value would be dynamically provided from the `{{product.id}}` expression in each iteration of the loop.

By using the `productId` parameter and the double curly bracket syntax, you can make this automation more flexible and reusable. Instead of creating separate automations for each product, you can have a single automation that can retrieve details for any product by simply providing the appropriate product ID as the parameter value from different sources, such as UI components or other automations.

Scenario: Handling optional parameters with fallback values

Let's consider a scenario where you have a "Task" entity with a required "Owner" column, but you want this field to be optional in the automation and provide a fallback value if the owner is not selected.

1. Create an automation with a parameter named `Owner` that maps to the `Owner` field of the Task entity.
2. Since the `Owner` field is required in the entity, the `Owner` parameter will synchronize with the required setting.
3. To make the `Owner` parameter optional in the automation, toggle the `required` setting off for this parameter.
4. In your automation logic, you can use an expression like `{{params.Owner || currentUser.userId}}`. This expression checks if the `Owner` parameter is provided. If it's not provided, it will fallback to the current user's ID as the owner.
5. This way, if the user doesn't select an owner in a form or component, the automation will automatically assign the current user as the owner for the task.

By toggling the `required` setting for the `Owner` parameter and using a fallback expression, you can decouple it from the entity field requirement, make it optional in the automation, and provide a default value when the parameter is not provided.

Defining automation parameter types

By using parameter types to specify data types and set requirements, you can control the inputs for your automations. This helps ensure your automations run reliably with the expected inputs.

Synchronizing types from an entity

Dynamically synchronizing parameter types and requirements from entity field definitions streamlines building automations that interact with entity data, ensuring that the parameter always reflects the latest entity field type and requirements.

The following procedure details general steps for synchronizing parameter types from an entity:

1. Create an entity with typed fields (e.g. Boolean, Number, etc.) and mark fields as needed.
2. Create a new automation.
3. Add parameters to the automation, and when choosing the **Type**, choose the entity field you want to sync with. The data type and required setting will automatically synchronize from the mapped entity field
4. If needed, you can override the "required" setting by toggling it on/off for each parameter. This means the required status will not be kept in sync with the entity field, but otherwise, it will remain synchronized.

Manually defining types

You can also define parameter types manually without synchronizing from an entity

By defining custom parameter types, you can create automations that accept specific input types and handle optional or required parameters as needed, without relying on entity field mappings.

1. Create an entity with typed fields (e.g. Boolean, Number, etc.) and mark fields as needed.
2. Create a new automation.
3. Add parameters to the automation, and when choosing the **Type**, choose desired type.

Configuring dynamic values to be passed to automation parameters

Once you've defined parameters for an automation, you can pass values to them when invoking the automation. You can pass parameter values in two ways:

1. **Component triggers:** If you're invoking the automation from a component trigger, such as a button click, you can use JavaScript expressions to pass values from the component context. For example, if you have a text input field named `emailInput`, you can pass its value to the `email` parameter with the following expression: `ui.emailInput.value`.
2. **Other automations:** If you're invoking the automation from another automation, you can use JavaScript expressions to pass values from the automation context. For example, you can pass the value of another parameter or the result of a previous action step.

Type safety

By defining parameters with specific data types, such as `String`, `Number`, or `Boolean`, you can ensure that the values passed into your automation are of the expected type.

Note

In App Studio, date(s) are ISO string dates, and those will be validated too.

This type safety helps prevent type mismatches, which can lead to errors or unexpected behavior in your automation logic. For example, if you define a parameter as a `Number`, you can be confident that any value passed to that parameter will be a number, and you won't have to perform additional type checks or conversions within your automation.

Validation

You can add validation rules to your parameters, ensuring that the values passed into your automation meet certain criteria.

While App Studio does not provide built-in validation settings for parameters, you can implement custom validations by adding a JavaScript action to your automation that throws an error if specific constraints are violated.

For entity fields, a subset of validation rules, such as minimum/maximum values, are supported. However, those are not validated at the automation level, only at the data layer, when running Create/Update/Delete Record actions.

Best practices for automation parameters

To ensure that your automation parameters are well-designed, maintainable, and easy to use, follow these best practices:

1. **Use descriptive parameter names:** Choose parameter names that clearly describe the purpose or context of the parameter.
2. **Provide parameter descriptions:** Take advantage of the **Description** field when defining parameters to explain their purpose, constraints, and expectations. These descriptions will be surfaced in the JSDoc comments when referencing the parameter, as well as in any user interfaces where users need to provide values for the parameters when invoking the automation.
3. **Use appropriate data types:** Carefully consider the data type of each parameter based on the expected input values, for example: String, Number, Boolean, Object.
4. **Validate parameter values:** Implement appropriate validation checks within your automation to ensure that parameter values meet specific requirements before proceeding with further actions.
5. **Use fallback or default values:** While App Studio does not currently support setting default values for parameters, you can implement fallback or default values when consuming the parameters in your automation logic. For example, you can use an expression like `{{ params.param1 || "default value" }}` to provide a default value if the `param1` parameter is not provided or has a false value.
6. **Maintain parameter consistency:** If you have multiple automations that require similar parameters, try to maintain consistency in parameter names and data types across those automations.
7. **Document parameter usage:** Maintain clear documentation for your automations, including descriptions of each parameter, its purpose, expected values, and any relevant examples or edge cases.
8. **Review and refactor frequently:** Periodically review your automations and their parameters, refactoring or consolidating parameters as needed to improve clarity, maintainability, and reusability.

9. **Limit the number of parameters:** While parameters provide flexibility, too many parameters can make an automation complex and difficult to use. Aim to strike a balance between flexibility and simplicity by limiting the number of parameters to only what is necessary.
10. **Consider parameter grouping:** If you find yourself defining multiple related parameters, consider grouping them into a single *Object* parameter.
11. **Separate concerns:** Avoid using a single parameter for multiple purposes or combining unrelated values into a single parameter. Each parameter should represent a distinct concern or piece of data.
12. **Use parameter aliases:** If you have parameters with long or complex names, consider using aliases or shorthand versions within the automation logic for better readability and maintainability.

By following these best practices, you can ensure that your automation parameters are well-designed, maintainable, and easy to use, ultimately improving the overall quality and efficiency of your automations.

Generative AI in App Studio

AWS App Studio provides integrated generative AI capabilities to accelerate development and streamline common tasks. You can leverage generative AI to generate apps, data models, sample data, configurations, and even get contextual help while building apps.

Generating your app

For an accelerated start, you can generate entire applications using natural language prompts powered by AI. This capability allows you to describe your desired app functionality, and GenAI will automatically build out the data models, user interfaces, workflows, and connectors.

Generating your data models

You can automatically generate an entity with fields, data types, and data actions based on the provided entity name. For more information about creating entities, including creating entities using GenAI, see [Creating an entity in an App Studio app](#).

You can also update an existing entity in the following ways:

- Add more fields to an entity. For more information, see [Adding, editing, or deleting entity fields](#).

- Add data actions to an entity. For more information, see [Creating data actions](#).

Generating sample data

You can generate sample data for your entities based on the entity's fields. This is useful to test your application before connecting external data sources, or testing your application in the Development environment, which doesn't communicate to external data sources. For more information, see [Adding or deleting sample data](#).

Once you publish your app to Testing or Production, your live data sources and connectors are used.

Configuring actions for AWS services

When integrating with AWS services like Amazon Simple Email Service, you can use AI to generate an example configuration with pre-populated fields based on the selected service. To try it out, in the **Properties** menu of an **Invoke AWS** automation action, expand the **Configuration** field by choosing the double-sided arrow. Then, choose **Generate sample configuration**.

Mocking responses

You can generate mocked responses for AWS service actions. This is helpful for testing your application in the Development environment, which doesn't communicate to external data sources.

Asking AI for help

Within the application studio, you'll find an **Ask AI for help** button. Use this to get contextual suggestions, documentation, and guidance related to the current view or selected component. Ask general questions about App Studio, app building best practices, or your specific application use case to receive tailored information and recommendations.

Using JavaScript to write expressions in App Studio

In AWS App Studio, you can use JavaScript expressions to dynamically control the behavior and appearance of your applications. Single-line JavaScript expressions are written within double curly braces, `{{ }}`, and can be used in various contexts such as automations, UI components, and data queries. These expressions are evaluated at runtime and can be used to perform calculations, manipulate data, and control application logic.

App Studio provides native support for three JavaScript open source libraries: Luxon, UUID, Lodash as well as SDK integrations to detect JavaScript syntax and type-checking errors within your app's configurations.

Basic syntax

JavaScript expressions can include variables, literals, operators, and function calls. Expressions are commonly used to perform calculations or evaluate conditions.

See the following examples:

- `{{ 2 + 3 }}` will evaluate to 5.
- `{{ "Hello, " + "World!" }}` will evaluate to "Hello, World!".
- `{{ Math.max(5, 10) }}` will evaluate to 10.
- `{{ Math.random() * 10 }}` returns a random number (with decimals) between [0-10).

Interpolation

You can also use JavaScript to interpolate dynamic values within static text. This is achieved by enclosing the JavaScript expression within double curly braces, like the following example:

```
Hello {{ currentUser.firstName }}, welcome to App Studio!
```

In this example, `currentUser.firstName` is a JavaScript expression that retrieves the first name of the current user, which is then dynamically inserted into the greeting message.

Concatenation

You can concatenate strings and variables using the `+` operator in JavaScript, as in the following example.

```
{{ currentRow.FirstName + " " + currentRow.LastName }}
```

This expression combines the values of `currentRow.FirstName` and `currentRow.LastName` with a space in between, resulting in the full name of the current row.

Date and time

JavaScript provides various functions and objects for working with dates and times. For example:

`{{ new Date().toLocaleDateString() }}` returns the current date in a localized format.

Code blocks

In addition to expressions, you can also write multi-line JavaScript code blocks. Unlike expressions, code blocks do not require curly braces. Instead, you can write your JavaScript code directly within the code block editor.

Note

While expressions are evaluated and their values are displayed, code blocks are run, and their output (if any) is displayed.

Global variables and functions

App Studio provides access to certain global variables and functions that can be used within your JavaScript expressions and code blocks. For example, `currentUser` is a global variable that represents the currently logged-in user, and you can access properties like `currentUser.role` to retrieve the user's role.

Accessing UI component values

One of the powerful features of App Studio is the ability to access values from UI components within expressions. This allows for dynamic behavior and data binding between components, which builders can use to create truly interactive and data-driven applications.

The `ui` namespace provides read-only access to the values and properties of UI components on the same page. By referencing a component's name, you can retrieve its value or perform operations based on its state.

The following list contains the syntax for using the `ui` namespace to access UI component values.

- `{{ui.textInputName.value}}`: Retrieve the value of a text input component named *textInputName*.
- `{{ui.formName.isValid}}`: Check if all fields in the form named *formName* are valid.
- `{{ui.tableName.currentRow.columnName}}`: Access the value of a specific column in the current row of a table component named *tableName*.

- `{{ui.tableName.selectedRows}}`: Retrieve the selected rows in a table component named *tableName*.

For example:

- `{{ui.myTextField.value}}` gives you the current value of a text input field named *myTextField*.
- `{{ui.userForm.isValid}}` will check if all fields in a form named *userForm* are valid.
- `{{ui.ordersTable.currentRow.orderTotal}}` will retrieve the value of the *orderTotal* column in the current row of a table component named *ordersTable*.

However, to update or manipulate the value of a component, you need to use `RunComponentAction`. Within expressions, component values are read-only, but you can trigger actions that modify their values. Here's an example of how you can update the value of a text input component named *myInput* using `RunComponentAction`:

```
RunComponentAction(ui.myInput, "setValue", "New Value")
```

In this example, the `RunComponentAction` step calls the `setValue` action on the *myInput* component, passing in the new value, *New Value*.

By combining the ability to read component values within expressions and update them using `RunComponentAction` steps, you can create dynamic and interactive user interfaces that respond to user input and data changes.

Note that the `ui` namespace will only show components on the current page, as components are scoped to their respective pages.

Additional examples

- `{{ui.inputText1.value.trim().length > 0}}`: Check if the value of the *inputText1* component, after trimming any leading or trailing whitespace, has a non-empty string. This can be useful for validating user input or enabling/disabling other components based on the input text field's value.
- `{{ui.multiSelect1.value.join(", ")}}`: For a multi-select component named *multiSelect1*, this expression converts the array of selected option values into a comma-

separated string. This can be helpful for displaying the selected options in a user-friendly format or passing the selections to another component or automation.

- `{{ui.multiSelect1.value.includes("option1")}}`: This expression checks if the value *option1* is included in the array of selected options for the *multiSelect1* component. It returns true if *option1* is selected, and false otherwise. This can be useful for conditionally rendering components or taking actions based on specific option selections.
- `{{new Date().toLocaleDateString()}}`: This expression gets the current date and converts it to a localized string representation based on the user's locale settings. It can be used to display the current date in a user-friendly format or to pre-fill date fields with the current date.
- `{{new Date().toISOString()}}`: This expression generates the current date in the ISO format ("2023-06-15T10:30:00.000Z"), which is the format expected by many entities and components. It can be used to pre-fill date fields with the current date or timestamp.
- `{{ui.s3Upload1.files.length > 0}}`: For an Amazon S3 file upload component named *s3Upload1*, this expression checks if any files have been uploaded by checking the length of the files array. It can be useful for enabling/disabling other components or actions based on whether files have been uploaded.
- `{{ui.s3Upload1.files.filter(file => file.type === "image/png").length}}`: This expression filters the list of uploaded files in the *s3Upload1* component to only include PNG image files, and returns the count of those files. This can be helpful for validating or displaying information about the types of files uploaded.

Working with table data

The `currentRow` and `ui.tableName.selectedRow` objects provide access to table data, allowing builders to perform operations and manipulations based on the current or selected row.

Note that `currentRow` and `ui.tableName.data` have different structures. The `currentRow` object is based on the column mappings configured for the table, while `ui.tableName.data` contains the raw data from the entity.

The following list contains the syntax for working with table data.

- `{{currentRow.columnName}}`: Retrieve the value of the *columnName* column for the current row in a table.

- `{{ui.tableName.selectedRow.columnMappingName}}`: Retrieve the value of the `columnMappingName` column for the selected row in the table named `tableName`.

See the following examples:

- `{{currentRow.firstName + ' ' + currentRow.lastNamecolumnMapping}}`: Concatenate values from multiple columns to create a new column in a table.
- `{{ { "Blocked": "#", "Delayed": "#", "On track": "#" } [currentRow.statuscolumnMapping] + " " + currentRow.statuscolumnMapping}}`: Customize the display value of a field within a table based on the stored status value.
- `{{currentRow.colName}}` or `{{currentRow["First Name"]}}` or `{{currentRow}}` or `{{ui.tableName.selectedRows[0]}}`: Pass the referenced row's context within a row action.
- `{{ui.tableName.selectedRows[0].columnMappingName}}`: Reference the selected row's column name from other components or expressions on the same page.

Accessing automations

Automations allow you to run server-side logic and operations in App Studio. You can use expressions to process data, generate dynamic values, and incorporate results from previous actions.

Accessing automation parameters

You can pass dynamic values from UI components and other automations into automations, making them reusable and flexible. This is done using automation parameters with the `params` namespace as follows:

`{{params.parameterName}}`: Reference a value passed into the automation from a UI component or other source. For example, `{{params.ID}}` would reference a parameter named `ID`.

Manipulating automation parameters

You can use JavaScript to manipulate automation parameters. See the following examples:

- `{{params.firstName}} {{params.lastName}}`: Concatenate values passed as parameters.

- `{{params.numberParam1 + params.numberParam2}}`: Add two number parameters.
- `{{params.valueProvided?.length > 0 ? params.valueProvided : 'Default'}}`: Check if a parameter is not null or undefined, and has a non-zero length. If true, use the provided value; otherwise, set a default value.
- `{{params.rootCause || "No root cause provided"}}`: If the `params.rootCause` parameter is false (null, undefined, or an empty string), use the provided default value.
- `{{Math.min(params.numberOfProducts, 100)}}`: Restrict the value of a parameter to a maximum value (in this case, 100).
- `{{ DateTime.fromISO(params.startDate).plus({ days: 7 }).toISO() }}`: If the `params.startDate` parameter is "2023-06-15T10:30:00.000Z", this expression will evaluate to "2023-06-22T10:30:00.000Z", which is the date one week after the start date.

Accessing automation results from a previous action

Automations allow application to run server-side logic and operations, such as querying databases, interacting with APIs, or performing data transformations. The `results` namespace provides access to the outputs and data returned by previous actions within the same automation. Note the following points about accessing automation results:

1. You can only access results of previous automation steps within the same automation.
2. If you have actions named `action1` and `action2` in that order, `action1` cannot reference any results, and `action2` can only access `results.action1`.
3. This also works in client-side actions. For example, if you have a button that triggers an automation using the `InvokeAutomation` action. You can then have a navigation step with a `Run If` condition like `results.myInvokeAutomation1.fileType === "pdf"` to navigate to a page with a PDF viewer if the automation indicates the file is a PDF.

The following list contains the syntax for accessing automation results from a previous action using the `results` namespace.

- `{{results.stepName.data}}`: Retrieve the data array from an automation step named `stepName`.
- `{{results.stepName.output}}`: Retrieve the output of an automation step named `stepName`.

The way you access the results of an automation step depends on the type of action and the data it returns. Different actions may return different properties or data structures. Here are some common examples:

- For a data action, you can access the returned data array using `results.stepName.data`.
- For an API call action, you may access the response body using `results.stepName.body`.
- For an Amazon S3 action, you may access the file content using `results.stepName.Body.transformToWebStream()`.

See the documentation for the specific action types you're using to understand the shape of the data they return and how to access it within the `results` namespace. The following list contains some examples

- `{{results.getDataStep.data.filter(row => row.status === "pending").length}}`: Assuming the *getDataStep* is an Invoke Data Action automation action that returns an array of data rows, this expression filters the data array to include only rows where the status field is equal to pending, and returns the length (count) of the filtered array. This can be useful for querying or processing data based on specific conditions.
- `{{params.email.split("@")[0]}}`: If the *email* parameter contains an email address, this expression splits the string at the @ symbol and returns the part before the @ symbol, effectively extracting the username portion of the email address.
- `{{new Date(params.timestamp * 1000)}}`: This expression takes a Unix timestamp parameter (*timestamp*) and converts it to a JavaScript Date object. It assumes that the timestamp is in seconds, so it multiplies it by 1000 to convert it to milliseconds, which is the format expected by the Date constructor. This can be useful for working with date and time values in automations.
- `{{results.stepName.Body}}`: For an Amazon S3 GetObject automation action named *stepName*, this expression retrieves the file content, which can be consumed by UI components like **Image** or **PDF Viewer** for displaying the retrieved file. Note that this expression would need to be configured in the **Automation output** of the automation to use in components.

Troubleshooting and debugging App Studio apps

Troubleshooting in the application studio

Using the debug panel

To assist with live debugging while you're building your apps, App Studio provides a collapsible builder debug panel that spans the pages, automations, and data tabs of the application studio. This panel shows both errors and warnings. While warnings serve as informative suggestions, such as resources that haven't been configured, errors must be resolved to successfully build your app. Each error or warning includes a **View** link which can be used to navigate to the location of the issue.

The debug panel automatically updates with new errors or warnings as they occur, and the errors or warnings automatically disappear once resolved. The state of error messages is persisted when you leave the builder.

Contextual JavaScript syntax feedback

App Studio features JavaScript error detection, highlighting errors by underlining your code with red lines. These compile errors, which will prevent the app from building successfully, indicate issues such as typos, invalid references, invalid operations, and incorrect outputs for required data types.

Troubleshooting while previewing an app

Within the interactive preview or in a live production app, a user needs to be able to easily understand the data flow of their app end-to-end better. This includes tracing parameters and seeing inputs/outputs of action steps, context helpful for debugging and broadly ensuring the correct operation of interactions.

Troubleshooting in the Testing environment

Using your browser console to debug

Since actions are not invoked while previewing your app, your app will need to be published to the Testing environment to test its call and response handling. If an error occurs during the execution of your automation or if you want to understand why the application behaves in certain way, you can use your browser's console for real time debugging.

To use your browser console to debug apps in the Testing environment

1. Append `?debug=true` to the end of the URL and press enter.
2. Open your browser console to start debugging by exploring your action or API inputs and outputs.
 - In Chrome: Right click in your browser and choose **Inspect**. For more information about debugging with Chrome DevTools, see the [Chrome DevTools documentation](#).
 - In Firefox: Press and hold or right-click on a webpage element, then choose **Inspect Element**. For more information about debugging with Firefox DevTools, see the [Firefox DevTools User Docs](#).

Debugging with logs from published apps in Amazon CloudWatch Logs

Amazon CloudWatch Logs monitors your AWS resources and the applications you run on AWS in real time. You can use CloudWatch Logs to collect and track metrics, which are variables you can measure for your resources and applications.

For debugging App Studio apps, CloudWatch Logs is useful for tracking errors that occur during an app's execution, auditing information, and providing context on user actions and proprietary interactions. The logs offer historical data, which you can use to audit application usage and access patterns, as well as review errors encountered by users.

Note

CloudWatch Logs does not provide real-time traces of parameter values passed from the UI of an application.

Use the following procedure to access logs from your App Studio apps in CloudWatch Logs.

1. In the App Studio application studio for your app, locate and note your app ID by looking at in the URL. The app ID may look something like this: `802a3bd6-ed4d-424c-9f6b-405aa42a62c5`.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Log groups**.

4. Here you will find five **Log Groups** per application. Depending on the type of information you are interested in, select a group and write a query for the data you want to discover.

The following list contains the log groups and information about when to use each:

1. `/aws/appstudio/teamId/appId/TEST/app`: Use to debug automation responses, query failures, component errors, or JavaScript code related to the version of your app currently published to the Testing environment.
 2. `/aws/appstudio/teamId/appId/TEST/audit`: Use to debug JavaScript code errors, such as conditional visibility or transformation, as well as login or permissions user errors related to the version of your app currently published to the Testing environment.
 3. `/aws/appstudio/teamId/setup`: Use to monitor builder or admin actions.
 4. `/aws/appstudio/teamId/appId/PRODUCTION/app`: Use to debug automation responses, query failures, component errors, or JavaScript code related to the version of your app currently published to the Production environment.
 5. `/aws/appstudio/teamId/appId/PRODUCTION/audit`: Use to debug JavaScript code errors, such as conditional visibility or transformation, as well as login or permissions user errors related to the version of your app currently published to the Production environment.
5. Once you are in a log group, you can either pick the most recent log streams, or one with a last event time closest to the time of interest, or you can choose to search all log streams to search across all events on that log group. For more information about viewing log data in CloudWatch Logs, see [View log data sent to CloudWatch Logs](#).

You can also use the **Logs Insights** in CloudWatch Logs query multiple log groups at once. For more information about CloudWatch Logs Insights, [Analyzing log data with CloudWatch Logs Insights](#) in the Amazon CloudWatch Logs User Guide.

Troubleshooting connector issues

This section contains some common troubleshooting guidance for connector issues. You must be a member of an admin group to view or edit connectors.

- Check the configuration of the resources in the product or service that your connector is connecting to. Some resources, such as Amazon Redshift tables, require additional configuration to use with App Studio.

- Check the IAM role you're using to connect to AWS services. Make sure the trust policy is configured to provide access to the account used to set up App Studio.
- Check your connector configuration. For AWS services, go to the connector in App Studio and ensure the correct Amazon Resource Name (ARN) is included and the AWS Region specified is the one that contains your resources.

Building an app with multiple users

Multiple users can work on a single App Studio app, however only one user can edit an app at one time. See the following sections to information about inviting other users to edit an app, and the behavior when multiple users try to edit an app at the same time.

Invite builders to edit an app

Use the following instructions to invite other builders to edit an App Studio app.

To invite other builders to edit an app

1. If necessary, navigate to the application studio of your application.
2. Choose **Share**.
3. In the **Development** tab, use the text box to search for and select groups or individual users that you want to invite to edit the app.
4. For each user or group, choose the dropdown and select the permissions to give to that user or group.
 - **Co-owner**: Co-owners have the same permissions as app owners.
 - **Edit only**: Users with the **Edit only** role have the same permissions as owners and co-owners, except for the following:
 - They cannot invite other users to edit the app.
 - They cannot publish the app to the Testing or Production environments.
 - They cannot add data sources to the app.
 - They cannot delete or duplicate the app.

Attempting to edit an app that is being edited by another user

A single App Studio app can only be edited by one user at a time. See the following example to understand what happens when multiple users try to edit an app at the same time.

In this example, User A is currently editing an app, and has shared it with User B. User B then attempts to edit the app that is being edited by User A.

When User B tries to edit the app, a dialog box will appear informing them that User A is currently editing the app, and that continuing will kick User A out of the application studio, and all changes will be saved. User B can choose to cancel and let User A continue, or continue and enter the application studio to edit the app. In this example, they choose to edit the app.

When User B chooses to edit the app, User A receives a notification that User B has started editing the app, and their session has ended. Note that if User A had the app open in an inactive browser tab, they may not receive the notification. In this case, if they try to come back to the app and try to make an edit, they will receive an error message and be guided to refresh the page, which will return them to the list of applications.

Security in AWS App Studio

Topics

- [Security considerations and mitigations](#)
- [Service-linked roles for App Studio](#)
- [AWS managed policies for AWS App Studio](#)

Security considerations and mitigations

Security considerations

When dealing with data connectors, data models, and published applications, several security concerns arise related to data exposure, access control, and potential vulnerabilities. The following list includes the primary security concerns.

Improper configuration of IAM roles

Incorrect configuration of IAM roles for data connectors can lead to unauthorized access and data leaks. Granting overly permissive access to a data connector's IAM role can allow unauthorized users to access and modify sensitive data.

Using IAM roles to perform data operations

Since end users of an App Studio app assume the IAM role provided in the connector configuration to perform actions, those end users might get access to data to which they typically do not have access.

Deleting data connectors of published applications

When a data connector is deleted, the associated secret credentials are not automatically removed from published applications that are already using that connector. In this scenario, if an application has been published with certain connectors, and one of those connectors is deleted from App Studio, the published application will continue to work using the previously stored connector credentials. It is important to note that the published app will remain unaffected and operational despite the connector deletion.

Editing data connectors on published applications

When a data connector is edited, the changes are not automatically reflected in published applications that are using that connector. If an application has been published with certain connectors, and one of those connectors is modified in App Studio, the published application will continue to use the previously stored connector configuration and credentials. To incorporate the updated connector changes, the application must be republished. Until the app is republished, it will remain incorrect and non-operational, or unaffected and operational but will not reflect the latest connector modifications.

Security risk mitigation recommendations

This section lists mitigation recommendations to avoid security risks detailed in the previous security considerations section.

1. **Proper IAM role configuration:** Ensure that IAM roles for data connectors are correctly configured with the principle of least privilege to prevent unauthorized access and data leaks.
2. **Restricted app access:** Only share your apps with users who are authorized to view or perform actions on the application data.
3. **App publishing:** Ensure that apps are republished whenever a connector is updated or deleted.

Service-linked roles for App Studio

App Studio uses a [service-linked role](#) named `AWSServiceRoleForAppStudio` for the permissions that it requires to call other AWS services on your behalf. A service-linked role is a unique type of AWS Identity and Access Management (IAM) role that is linked directly to an AWS service, in this case, App Studio. The service-linked role provides a secure way to delegate permissions to App Studio because only App Studio can assume the service-linked role.

App Studio uses the service-linked role to persistently manage AWS services, to maintain the application building experience.

The service-linked role makes setting up App Studio easier because you don't have to manually add necessary permissions. App Studio defines the permissions of its service-linked role, and unless the permissions are defined otherwise, only App Studio can assume the role. The defined permissions include the trust policy and the permissions policy, and you can't attach that permissions policy to any other IAM entity.

Contents

- [Service-linked role permissions for App Studio](#)
- [Creating a service-linked role for App Studio](#)
- [Editing a service-linked role for App Studio](#)
- [Deleting a service-linked role for App Studio](#)

Service-linked role permissions for App Studio

App Studio uses the service-linked role named `AWSServiceRoleForAppStudio`. It's a service-linked role required for App Studio to persistently manage AWS services, to maintain the application building experience.

The `AWSServiceRoleForAppStudio` service-linked role uses the following trust policy, and only trust the `appstudio-service.amazonaws.com` service.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appstudio-service.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For permissions, the `AWSServiceRoleForAppStudio` service-linked role uses the `AppStudioServiceRolePolicy` managed policy. For more information about the managed policy, including the permissions it includes, see [AWS managed policy: AppStudioServiceRolePolicy](#).

Creating a service-linked role for App Studio

The `AWSServiceRoleForAppStudio` service-linked role is automatically created when a user requests a specific operation.

Editing a service-linked role for App Studio

App Studio doesn't allow you to edit the `AWSServiceRoleForAppStudio` service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role by using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for App Studio

If you no longer need to use App Studio, we recommend that you delete the service-linked role. That way, you don't have an unused entity that isn't actively monitored or maintained.

To manually delete the service-linked role using IAM

Use the IAM console, the IAM CLI, or the IAM API to delete the `AWSServiceRoleForAppStudio` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

AWS managed policies for AWS App Studio

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

AWS managed policy: AppStudioServiceRolePolicy

You can't attach AppStudioServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows App Studio to perform actions on your behalf. For more information, see [Service-linked roles for App Studio](#).

This policy grants permissions that allow the service-linked role to manage AWS resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AppStudioResourcePermissionsForCloudWatch",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/appstudio/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
      }
    },
    {
      "Sid": "AppStudioResourcePermissionsForSecretsManager",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager>DeleteSecret",
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecret",
        "secretsmanager:TagResource"
      ],
    }
  ]
}
```

```

    "Resource": "arn:aws:secretsmanager:*:*:secret:appstudio-*",
    "Condition": {
      "ForAllValues:StringEquals": {
        "aws:TagKeys": [
          "IsAppStudioSecret"
        ]
      },
      "StringEquals": {
        "aws:ResourceAccount": "${aws:PrincipalAccount}",
        "aws:ResourceTag/IsAppStudioSecret": "true"
      }
    }
  },
  {
    "Sid": "AppStudioResourcePermissionsForSSO",
    "Effect": "Allow",
    "Action": [
      "sso:GetManagedApplicationInstance",
      "sso-directory:DescribeUsers",
      "sso-directory:ListMembersInGroup"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceAccount": "${aws:PrincipalAccount}"
      }
    }
  }
]
}

```

App Studio updates to AWS managed policies

View details about updates to AWS managed policies for App Studio since this service began tracking these changes.

Change	Description	Date
App Studio started tracking changes	App Studio started tracking changes for its AWS managed policies.	June 28, 2024

Document history for the AWS App Studio User Guide

The following table describes the documentation releases for AWS App Studio.

Change	Description	Date
Updated topics: Previewing, publishing, and sharing App Studio apps	Expanded the previewing, publishing, and sharing documentation to add clarity, match the experience in the service, and provide additional information around the publishing environments and viewing apps in them. For more information, see Previewing, publishing, and sharing applications .	August 2, 2024
New topic: Building an app with multiple users	Expanded the previewing, publishing, and sharing documentation to add clarity, match the experience in the service, and provide additional information around the publishing environments and viewing apps in them. For more information, see Building an app with multiple users .	August 2, 2024
Updated topic: Connecting App Studio to AWS services	Added information about creating and providing IAM roles for providing access to AWS resources when creating an Other AWS services connector. For more information, see Connect to AWS	July 29, 2024

services using the Other AWS services connector.		
Updated topic: Add instructions for creating an AWS administrative user as part of setting up	Added instructions in the setting up App Studio documentation to create an administrative user for managing AWS resources. Also made updates throughout the connector documentation to recommend using that user.	July 24, 2024
New topic: Connect to Amazon Bedrock	Added a topic with instructions for creating a connector for Amazon Bedrock. Builders can use the connector to build apps that use Amazon Bedrock. For more information, see Connect to Amazon Bedrock .	July 24, 2024
Initial release	Initial release of the AWS App Studio User Guide	July 10, 2024